Static Allocation of Computation to Processors in Multicomputers

Michael G. Norman

PhD

University of Edinburgh

1993

To Ruth Thomas

Acknowledgements

I would like to thank Peter Thanisch, my supervisor, for sound guidance and extremely consciencious supervision of my PhD. His hard work and personal interest was invaluable, and way beyond that which one expects of a supervisor. I would also like to thank Professor David Wallace, my second supervisor and Director of the Edinburgh Parallel Computing Centre, and Professor Roland Ibbett, Head of the Department of Computer Science and Associate Director for Research in the EPCC.

As indicated in the declaration much of the work was done in collaboration with others. I would like to thank George Chochia, Tim Kempster, Susanna Pelagatti, Cristina Boeres, Gary Chang and Emmanuel Issman.

The work was also supported by more general collaboration with others in EPCC and the Department of Computer Science. In particular I would like to mention Tim Harris, Brian Wylie, Lyndon Clarke, Tommy Kelly, Neil Macdonald and Greg Wilson.

I would like to thank my family, and finally to thank Ruth for proof reading and putting up with the stress of my doing the PhD.

Abstract

In this thesis we address the static mapping problem—that is the problem of allocating computation to processors—in a MIMD, distributed-memory architecture: *a multicomputer*. We are primarily interested in the way in which the computation and the multicomputer can be modelled: the features of the multicomputer and the computation that are included and left out, and the way in which that impacts upon the predictions made by the models for the performance of computations. We try to put the various published formulations of the mapping problem into the context of the multicomputer, and to identify correspondences between features of the models underlying the formulations, and features of the multicomputer and the computation.

The two types of models which we choose to consider in detail are precedence constrained scheduling with interprocessor communication delay, and static process based models. We review approaches to hybridising the two types of model and propose such a model of our own. We also consider the impact of message contention in the multicomputer.

We analyse the models underlying formulations of the mapping problem in a number of ways. We look at the way in which performance gains can be achieved by adding more processors to the models. We consider the way in which the complexity of mapping problems depends upon the modelling of interprocessor communication. We compare bounds on performance given for approximation algorithms in different, but related models. We show, for an example computation, how the predictions of the various models differ and

how these differences might lead the multicomputer programmer to different conclusions.

Finally we relate the predicted performance in some of the models of our example computation with that observed when executing it on a real multicomputer.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

This thesis is about the problem of mapping computation to processors in a parallel computer in order to optimise performance. We choose to consider a restricted class of such problems, one which we argue is important to the programmer of the class of parallel computers known as *multicomputers*. In this thesis we bring forward evidence to support the following claim.

> Progress in solving the problem of allocating computation to processors in multicomputers is hindered by the lack of appropriate architectural and computational models rather than by the complexity or non-approximability of the various existing formulations of the problem.

1.1.1 Area of Interest and Novelty of Approach

The general problem of allocating computation to processors in a parallel computer is well studied, and we certainly do not aim to cover the whole of the field. The problems we will consider are restricted in both the type of computation and the type of parallel computer architecture.

The class of *architectures* we consider in detail is the distributed memory multiple-instruction multiple-data computer: the *multicomputer* of which the various commercial hypercube systems and transputer based machines are examples. They consist of processors[1], each of which has its own memory. There is no global memory and processors communicate by message passing. The

[1]Some authors refer to them as *processing elements* since they may consist of more than just a simple processing unit.

processors in a multicomputer may execute asynchronously. Informally, the processors need not all be on the same chip, but should be in the same box; that is, we are interested in *parallel* systems, not *distributed* systems.

The types of *computation* we consider are those for which a static "all-knowing" representation (usually graph based) can be derived in advance of program execution. Although we will often assume the existence of such representations and it may be possible to derive them for the regular computations discussed in the later chapters of the thesis, it must be understood that they are rarely readily available for real computations running on real multicomputers. In particular such representations may be program-data dependent.

Our treatment of the subject is novel in that we are not primarily interested in defining algorithms for mapping, but in the assumptions inherent in the models underlying the mapping problem. Both the multicomputer and the computation must be modelled in some way. We are interested in the way in which such models are formulated: in putting into the context of the multicomputer the assumptions inherent in such formulations; in the way in which such assumptions affect the complexity and approximability of mapping problems; and in the way in which different assumptions lead to different predictions of performance. We are also, ultimately, interested in the predictive power of such models for mapping problems encountered by multicomputer programmers.

As an example of our treatment, consider the following. There are a number of algorithms for mapping computation which have performance guarantees. That is they guarantee to find a mapping which performs at worst, say, half as well as an optimal mapping. We show that such guarantees of performance are only as good as the underlying models. If a given computation and multicomputer differ from the underlying model, then the performance guarantee will not apply to that computation running on that multicomputer. Moreover, another algorithm, possibly with a weaker "performance guarantee" might well give better results in practice.

The nature of the thesis is largely analytical: we derive features of models in the abstract, such as the complexity of mapping problems formulated in the models, and bounds on the number of processors that can be used in the models by mapping algorithms. We also analyse the performance of particular mapping algorithms for particular models. In addition we describe a range of models in terms of a single framework, which may be thought of as another model, but one which has a strong relationship to the multicomputer and to multicomputer computations. Finally, towards the end of the thesis we take a more experimental approach in validating the predictions of models, including one proposed in the thesis, against the performance of a real computation running on a real multicomputer.

1.1.2 Thesis Structure

The structure of the thesis reflects the differences between the models under consideration. In particular there is no *single* literature survey. Where models incorporate markedly different features they are considered in different chapters, and existing work is surveyed in many chapters, including this one.

This chapter sets the context of our work and outlines the underlying literature. For example it explains the difference between the task based scheduling models discussed in Chapters 2, 3 and 4, and the process based models discussed in Chapters 5 and 6. It also defines the multicomputer-based framework in which models are presented, and discusses the influence of the multicomputer's interprocessor communication network. Finally, it reviews some relevant literature and some models which are not covered in any detail in later chapters.

Chapter 2 gives an overview of the various *scheduling* models that have been presented in the past. It explains how they relate to each other and develops some notation within which they can all be expressed.

Chapter 3 considers various scheduling models from three different viewpoints. First we analyse the way in which performance gains can be achieved by adding more processors to the model. Second we review and derive complexity results for scheduling in such models. Third we derive the relationship between the scheduling models and the framework defined in this chapter.

Chapter 4 considers algorithms that have been proposed for scheduling models. In particular we show how the performance guarantee of a particular heuristic algorithm relies on the assumption in the underlying model of the availability of an unlimited numbers of processors. We show the NP completeness of remapping its output to achieve comparable performance guarantees on a fixed number of processors, and propose a related algorithm for which the performance guarantee depends upon the number of processors used. Finally in Chapter 4 we compare a pair of heuristic algorithms on the small range of models to which they both apply.

Chapter 5 is primarily a literature review. It describes a pair of process based models and puts them into the framework, and develops some notation for such models which is used in later chapters.

Chapter 6 considers the influence of the multicomputer network. First, we show the NP completeness of mapping in a process based formulation where it is the network of the multicomputer that is to be optimised. Mapping in a similar model with a fixed multicomputer network is not known to be NP complete. Second we show the NP completeness of a scheduling problem (which otherwise has a polynomial time algorithm) when the communications network is explicitly modelled.

Chapter 7 considers models which may be thought of as merging the process based models with the scheduling models. We review the relevant literature and propose a model of our own which will be used in subsequent chapters.

Chapters 8 and 9 concern one particular computation: the diamond dag

which corresponds to, for example, the longest common subsequence computation which is used in gene sequence matching. In Chapter 8 we derive for a range of models the predicted performance of the diamond dag in a number of mappings. In Chapter 9 we test our predictions against the performance of a real computation running on a Meiko Computing Surface.

In our concluding remarks we attempt to identify recurring themes in the results of our analysis and experimentation, and to identify areas which may prove fruitful for further research.

1.1.3 Issues Not Considered in Detail

By restricting our discussion to mapping in multicomputers we are excluding a large number of related areas. For example, we ignore scheduling for VLIW architectures and hardware data-flow architectures (e.g. McDowell and Appelbe [1986], Gaudiot *et al.* [1988], and the complexity results of Fellows and Langston [1988]).

We also exclude multicomputers with special-purpose hardware, such as Barrier MIMD architectures [Dietz *et al.* 1992], which keep the processors in synchronisation at barrier points in the computation, thereby allowing programming techniques akin to those used for VLIW architectures.

We are not directly interested in the issues in scheduling pipelined or vector processors, as reviewed recently by Krishnamurthy [1990]; and we are not interested in issues of language design for multicomputers (see Bal *et al.* [1989]).

We ignore the interactions of multicomputer programs with operating system facilities such as those discussed by Chu and Lan [1987].

Casavant and Kuhl [1988] propose a taxonomy for describing mapping strategies which is determined by the types of algorithms being used. In their terms, our discussion is restricted to the *static* mapping problem, which allows

us to concentrate on the underlying modelling issues, rather than, say, the problems of deriving parameters of computations or the reliability of local mapping strategies given only local information.

1.2 Formulating the Mapping Problem

Multicomputers can have a performance-price ratio that is vastly superior to that of uniprocessors. Having said that, it is a much harder intellectual exercise to develop efficient software that can extract this performance from a multicomputer. Each stage of the conventional software development cycle - algorithm design, implementation, debugging, performance analysis and tuning - is more complex when the target hardware is a multicomputer.

Much of the extra difficulty, however, is caused by components in the software development cycle that have no counterparts in the uniprocessor case. In particular, the computation must be designed in such a way that it can be divided into modules which can be assigned to the various processors in the multicomputer. Furthermore, it may also be necessary to devise a schedule for each processing element, i.e. an order in which the modules of computation assigned to a particular processing element are to be executed.

In this thesis, we refer to the problem of finding such an assignment - and, where appropriate, a schedule - as a *mapping problem*[2].

In broad terms we can consider three aspects to the mapping problem which we require to model: the processors and their communications facilities; the

[2]In the literature on parallel computing, there is no standard definition of "mapping". For example, some other authors use the term to mean just an assignment.

modules and their communications patterns; the function which is used to determine the *cost* of a mapping. It is worth stressing at this stage that much of the work we consider was originally formulated in the context of *Operations Research* or *Distributed Systems*, and thus our finding it inapplicable to the problem of mapping in multicomputers is not in any way a criticism of the work.

1.2.1 Modelling Costs

There are several different ways to formulate "optimisation" in the mapping problem. For example, for software such as operating systems, it is useful to construe the optimisation in terms of maximising the throughput of jobs. In a real time system design problem, the designer may be interested in the minimum number of processors that can guarantee a particular level of performance. Alternatively, the number of processors in the multicomputer may be fixed, and the performance of the software may be optimised with respect to a single program. The throughput based formulation has been used by a number of authors e.g. Baccelli and Liu [1990] and Bokhari [1981b]; the problem of finding a minimum number of processors has been addressed by others such as Fernández and Bussel [1973], Al-Mouhammed [1990] and Houstis [1990]. The main emphasis in this thesis is on the formulation of the problem for a fixed number of processors which are executing a single program. By focussing our attention on the execution of a single program, the optimisation of the mapping becomes the minimisation of *time to completion*, which is the elapsed wall-clock time between the moment when the multicomputer starts to execute the program to the moment at which the result is presented.

1.2.2 Modelling Processors

We shall often refer to a set P of processors. It is common (e.g. Graham *et al.* [1979]) to consider three ways in which the processors making up a parallel computer can vary in processing speed. They may be *identical,* that is every processor processes all modules at the same speed as every other. They may be *uniform,* that is the time of any given processor to process any module is a constant integer multiple of a time which is a parameter of that module. Alternatively they may be *unrelated,* for example a processor p could be *faster* than processor q at computing module γ, but *slower* than q at computing some other module δ. Since we are considering the *multicomputer* we shall be interested mainly in *identical* processors. The more general case of heterogeneous multicomputers corresponds to models of *unrelated* processors.

In order to model communications facilities between processors we often introduce a relationship between the *cost* of communication between modules and the processors to which modules are mapped.

There are a number of options:

- The cost of communication between modules is independent of the processors to which modules are mapped.

- The cost of communication between modules depends only upon whether or not they have been assigned to the same processor.

- The cost of communication between modules depends upon the processors to which they have been mapped. For example, processors may be considered to be connected in an undirected graph.

1.2.3 Modelling Modules

We shall often refer to a set Γ of modules that make up the computation. A module is a unit of computation that is executed sequentially. Modules can be executed preemptively or non-preemptively: that is they may or may not be allowed to be suspended. Modules are often algorithmic units – perhaps functions in a functional decomposition. Alternatively in data parallel applications they may correspond to the computation associated with divisions of a data space.

There are three basic types of models. Firstly there are models where no communication occurs between modules. Secondly there are models which consist of modules (referred to as *tasks*) arranged in *directed* acyclic graphs where an arc between a pair of tasks corresponds to both a precedence relationship and an associated communication event. Thirdly there are models which consist of modules (referred to as *processes*) arranged in undirected graphs where an arc corresponds to a volume of communication between processes. The directed graph models are often used by researchers interested in scheduling problems, and tend to have a basis in shared-memory multiprocessors whereas the undirected graph models tend to have a basis in distributed systems and are often used by researchers who are mapping programs which are explicitly parallel.

Although very different techniques are used for mapping in directed and undirected graph based models, these models may simply embody different views of the same computation. There is a sense in which the undirected graph models correspond to directed graph models in which some tasks has been predefined to be mapped as a single unit. Indeed there are mapping techniques for directed graphs which are based upon this approach [Kruatrachue and Lewis 1988].

1.3 A Framework for Discussing Mapping Problems

In this section we present a framework for discussing the mapping problem in terms of the activity of the processors of a multicomputer. We consider the case of a calculation – that is a set of algorithms and a set of data upon which the algorithms are to be applied – which is partitioned into a set of modules which are executed on a set, say P, of identical[3] processors. The framework bears some resemblance to the models of parallel computers described by Fox *et al.* [1988], or by Reed and Fujimoto [1987]. It differs in that it is more explicit in dealing only with the state of processors.

1.3.1 The Four Basic States

It is assumed that at all times during a given execution of a parallel program any processor $p \in P$ can be uniquely identified as being in one of four states:

- Performing the computation required by the calculation
- Performing computation associated with message transfer
- Performing computation associated with housekeeping operations
- Idle

We shall refer to the time that a processor p spends in these states as $T_{Calc}(p)$, $T_{Comm}(p)$, $T_{House}(p)$ and $T_{Idle}(p)$ respectively. We also define:

$$T_{Calc} = \sum_{p \in P} T_{Calc}(p)$$

and similarly for all other values which are subscripts in T.

---

[3]Recall our definition in Section 1.2.2.

1.3.2 Time Spent Calculating

We can define T_{Calc} to be a property of P and the calculation being performed, and completely independent of the partitioning of the calculation into modules and the mapping of modules to processors. Where the set of modules includes some re-calculation this will appear as a component of T_{House}.

1.3.3 Time Spent Communicating

We can sub-divide the term $T_{Comm}(p)$ as follows:

$$T_{Comm}(p) = T_{Init}(p) + T_{Term}(p) + T_{Route}(p),$$

where $T_{Init}(p)$, $T_{Term}(p)$ and $T_{Route}(p)$ are the amounts of time that a processor p spends performing the computation associated with message initiation, termination and through-routing respectively. Where processor p sends a message to processor q, the processing associated with communication contributes to $T_{Init}(p)$ and $T_{Term}(q)$, and if, for example, through-routing of the message causes costs to be incurred at processors other than p and q then it appears in T_{Route} on those processors[4]. It should be noted that the time a message may take to arrive is not solely dependent upon these factors.

T_{Term} and T_{Init} include all the computation which results from the partitioned address space: determining whether a message transfer is required; generating a packet; function call overhead associated with transfer and receipt of the packet and with generating any associated protocol; copying into and out of local address space; and stripping of packet headers.

---

[4]The above definition applies for multicast messages if they are considered to be multiple message transfers.

T_{Route}, the computation associated with intermediate node transfer of messages in a multicomputer, is seen only on such machines as first generation hypercubes and transputer based multicomputers running message-passing systems. It is envisaged that this overhead will disappear as the computation is taken over by dedicated hardware. This is not to say that the influence of the underlying processor network disappears, since it continues to show up in T_{House} and T_{Idle}.

1.3.4 Time Spent Housekeeping

There are a number of overheads associated with parallel computation which are neither present in the sequential case nor directly associated with communication. For completeness we include these as "housekeeping overheads", T_{House}, of which we identify three components.

$$T_{House}(p) = T_{Sched}(p) + T_{Inter}(p) + T_{Recalc}(p).$$

$T_{Recalc}(p)$, which of the three is the component we will come across most, is the overhead associated with recomputation of parts of the calculation. As we will see in Section 3.2 below, in some models this is worthwhile, and in others it is not. The computation associated with the calculation may appear only once in T_{Calc}. Recomputation must appear in T_{Recalc}. If calculation is performed more than once it is useful to consider the first computation to be *calculation*, the others *recalculation*. If it is initiated simultaneously on more than one processor then some arbitrary assignment of the computation events to T_{Calc} and T_{Recalc} must be made.

The other two overheads feature little in the rest of the thesis since they are rarely modelled by those researching into the mapping problem. They incorporate all computation that occurs in the parallel case that would not occur

in the serial case, and that cannot be attributed to initiating and terminating communication. $T_{Sched}(p)$ contains the overheads of local dynamic scheduling of computation where processor p is assigned more than one module. $T_{Inter}(p)$ contains all other such overheads.

1.3.5 Time Spent Idle

$T_{Idle}(p)$, the time processor p spends idle, is also partitioned:

$$T_{Idle}(p) = T_{Wait}(p) + T_{Fin}(p).$$

T_{Fin} is the time processors spend idle, having completed all their modules. It can be thought of as a place-filler. There is no cost associated with the extension of processing into T_{Fin}.

$T_{Wait}(p)$, the time that processor p spends idle with none of its modules executing, before it has performed all the computation associated with its modules, is a property of the global mapping of modules, and can be regarded as the overhead associated with load imbalance. Again there is no cost directly associated with extension of processing into $T_{Wait}(p)$ and so, for example, $T_{Recalc}(p)$ can be extended at the expense of $T_{Wait}(p)$, corresponding to re-computation of values so as to minimise overall program completion time. It is often useful for $T_{Wait}(p)$ to be considered the sum of two components:

$$T_{Wait}(p) = T_{Wait_S}(p) + T_{Wait_D}(p)$$

where $T_{Wait_S}(p)$ is the time processor p spends waiting for messages before they are sent, and $T_{Wait_D}(p)$ is the time that processor p spends waiting for messages that are in transit. If p's next scheduled task is awaiting more than one message from tasks on other processors then the time contributes to $T_{Wait_D}(p)$ only if *all* outstanding messages are in transit.

1.3.6 Interprocessor Communications

The above "processor-centric" framework does not consider the processor network which mediates the interprocessor communication in the multicomputer. This network can often be represented as an undirected graph – such as a hypercube or a torus. In its general form the network consists of routing elements (vertices) with channels (edges) between them. Messages are passed between adjacent routing elements, being *through-routed* as necessary. The processors are connected to individual routing elements in the network. In a number of so-called *direct* networks, there is a one-to-one correspondence between processors and routing elements.

Even amongst those corresponding to the same graph, different interprocessor communication networks have different performance characteristics. Those which are simplest to model are *packet switched* whereby messages are sent in one or more packets each of which moves independently through a network being stored at intermediate routing elements before being forwarded to its subsequent destination. In *circuit switched* and *wormhole* networks messages are transferred as sequences of *flits* which do not move independently. Circuit switched and wormhole systems differ in the way in which the sequence of channels through which a message is transferred is set up. There is a variation on the above which is used to avoid deadlock [Dally and Seitz 1987] and also to enhance performance [Duato 1992] , whereby channels are split into many virtual channels over which flits can move independently.

The communications network is often described as having *quiet* and *busy* performance characteristics. Quiet networks are relatively well understood; busy networks less so. Busy networks are discussed in Chapter 6. In this section, and in the majority of the thesis we consider only quiet networks, by which we are making the assumption that communication between all pairs of modules in the multicomputer program occurs across hardware which is not shared with

communications between other pairs of modules, at least for the duration of any communication. In these cases it is often possible to approximate the delay associated with message transfer in a multicomputer by a simple function of the number of interprocessor network links traversed, the number of words being transferred in the message and hardware specific constants.

This model is most appropriate for modelling simple store-and-forward message passing systems such as were written in software on first generation commercial hypercube machines and on transputer based machines. Circuit switched and wormhole systems are less adequately modelled by this approach. The constants must be replaced with functions which are derived experimentally from the machine. For example in the Intel iPSC/2 there is a strong discontinuity in performance at a message length of 100 bytes [Bomans and Roose 1989]. Nevertheless it is often possible to derive expressions which accurately model the communication delay. A number of studies have been published (see for example [Bokhari 1990] and [Bomans and Roose 1989]). See Seitz [1990] and Dally [1990a] for more discussion of message latencies in packet switched and circuit switched networks.

It should be stressed that there is no strong causal link between the components of T_{Comm} and the properties of the communication network. Delays in communication need not show up as overheads on any processor, and communications overheads on processors need not show up as delays to messages. We discuss this in more detail in Chapter 9 in the context of a particular computation executing on a particular multicomputer.

Chapter 2

An Overview of Scheduling Models

This chapter consists of four sections. The first three introduce a number of related scheduling models which might have predictive power for multicomputer performance. The fourth introduces a notation which is used in subsequent chapters.

The *No Communication* scheduling model of Section 2.1 and the *No Delay* scheduling model outlined in Section 2.2 below, correspond to the formulations that are the basis of scheduling theory, which developed from the 1950s onwards, and have been reviewed on a number of occasions (Graham *et al.* [1979], Conway *et al.* [1967], Coffman [1976] (including the chapter by Sethi), Gonzalez [1977], Blazewicz *et al.*[1991]), and still are of current interest (e.g. Ramamithram *et al.* [1990]). Although the complexity results and algorithms for such models underly much of what is discussed in the later chapters, it is not our purpose to discuss them in detail except where they relate to multicomputer systems.

A number of generalisations of these scheduling models have been proposed which might have relevance to the multicomputer programmer but which we only mention here in passing. In particular some authors (e.g. Kafura and Shen [1977] and Garey and Johnson [1975]) have extended scheduling models to allow constraints upon allocations of tasks so that they compete for resources, thereby modelling the situation in multicomputers which do not allow virtual memory and have localised software interfaces and special purpose hardware. In addition, a number of authors (e.g. Lo [1992], Blazewicz *et al.* [1986][1984], Wang and Cheng [1992], Du and Leung [1989], Nicol *et al.* [1992]) have recently considered the mapping problem for tasks which are themselves parallel programs. The models correspond closely to the subcube allocation problem for hypercubes studied by Chen and Lai [1988] and Chen and Shin [1987].

For the purposes of this thesis, however, we restrict our consideration of precedence constrained scheduling to models with the following properties:

- Identical processors

- Uniprocessor tasks
- Arbitrary positive integer task execution times
- No preemption
- Arbitrary non-negative integer interprocessor communication delays.

2.1 No Communication

The simplest models of parallel computation are those where computations are modelled as tasks, each of which is executed sequentially on a single processor and between which there is no communication. The problem is often referred to as a scheduling problem, but in truth, the latter assumption means that the tasks can be executed in any order, and the problem is simply one of balancing the load on the various processors to which the computation is being mapped.

The model corresponds to the execution of independent programs, say for example by a parallel batch server, in a way that multicomputer programmers often refer to as *embarrassingly parallel* (e.g. Fox *et al.* [1988]) or *event parallel* (e.g. Pritchard *et al.* [1987]). For example, in parallel implementations of ray-tracing algorithms, the computation is often decomposed into tasks corresponding to the computation of square tiles which together make up a screen. The tasks may be executed independently, and the computation time of the tasks varies. As long as the communications overheads associated with startup and termination of tasks can be ignored, dynamic mapping problems framed in the model may correspond accurately to the problem of predicting the performance of a task farming implementation of ray-tracing. Static versions of the mapping problem correspond to the problem of allocating such tasks according to a pre-planned strategy.

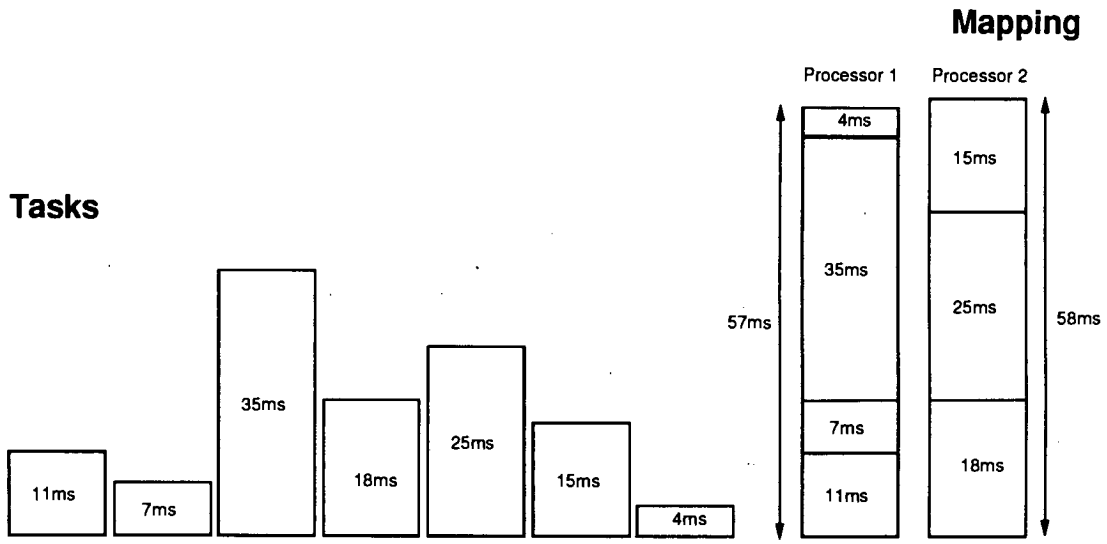


Figure 2–1: An Example Mapping Problem in a *No Communication* Scheduling Model and a Solution

As an example of a mapping problem of this type, the tasks in Figure 2–1 have execution times as shown and are allocated to two processors as indicated. The resulting execution time of the computation is 58 milliseconds. One of the processors is active for 57 milliseconds the other for 58 milliseconds and so the schedule is optimal amongst non-preemptive schedules. No schedule could have both processors active for 57.5 milliseconds without a preemption in which a task is executed on one processor, its execution terminated, and then restarted on another processor.

2.2 Tasks with Precedence

It might be that multicomputer programs with inter-task communications are not handled with sufficient accuracy by the model outlined in Section 2.1. In this section we describe a model of non-preemptive scheduling where again the program is represented as a set of tasks but this time the tasks communicate results to other tasks on termination, and the structure of the computation is

represented as a directed task graph in which a directed edge connects a pair of distinct tasks if and only if the task at the start of the edge requires the results of computation from the task at the end of the edge.

The directed task graph represents a partial order for an agenda of activities. It may be, for example, that at two or more separate stages in the computation it is necessary to perform a given computation—for example a sort operation on two different sets of data. This would be represented by two separate tasks/nodes in the task digraph. Consequently it is clear that in a valid task graph, there cannot exist a cycle in which a given task requires the results of a task to which it, in turn (either directly or indirectly) supplies results. The task graph is always a *directed acyclic graph* or dag.

Most researchers in this area of the mapping problem would, as for the model discussed in Section 2.1, expect the nodes to be labelled with integers representing the time required to execute them. The mapping problem, which is usually referred to as *Precedence Constrained Scheduling*, is now the problem of assigning tasks to processors and giving the processors local schedules for their tasks, subject to three constraints:

1. All tasks are executed. (This property is formalised below as the *Completeness* property of Definition 2.30.)

2. No processor is executing more than one task at a time (that is, it is not *Overbooked* as stated in Definition 2.31 below.)

3. The tasks that are defined to precede a given task have finished executing before that task is started (that is, it is not *Temporally Compromised* as stated in Definition 2.35 below.)

This definition of what constitutes a valid schedule is identical in all but the symbols, to that used by, for example, Papadimitriou and Yannakakis [1990],

or by Chretienne and Picouleau [1992], but is different from and more complex than that which has usually been used in the past, e.g. in the definition of Precedence Constrained Scheduling in Garey and Johnson [1979]. The added notational burden results from the fact that we wish to use the same definition in the models described in Section 2.3 below, where tasks may usefully be executed more than once. See Section 3.2 below.

Let us consider an example task graph (shown in Figure 2–2). This contains the same tasks as we discussed in the previous section, but imposes a precedence relation on the tasks. There is a single sink node corresponding to the task that presents the final result to the user, and a single source node corresponding to the task that inputs the user's request to start the computation.

In this case, we may wish to identify a *critical path* in the task graph, ie. a path from source to sink such that the sum of the node weights is maximised. The significance of the critical path is that, even assuming an unbounded number of processors, the sum of the execution times labelling the nodes on the critical path represents the minimum achievable time to completion for this task graph (assuming the program's designer got the labels correct). This contrasts with the model outlined in Section 2.1 above, where it is possible to use simultaneously as many processors as there are tasks.

The critical path has been identified on Figure 2–2, and corresponds to the shaded nodes. We can consider mapping the tasks to two processors as shown in Figure 2–3. Here we show the activity of the processors, through time, and also show communication events as arrows, and the buffering of messages that are received. Since we have mapped the critical path and only the critical path to one of the two processors, the time to completion of the schedule is minimal.

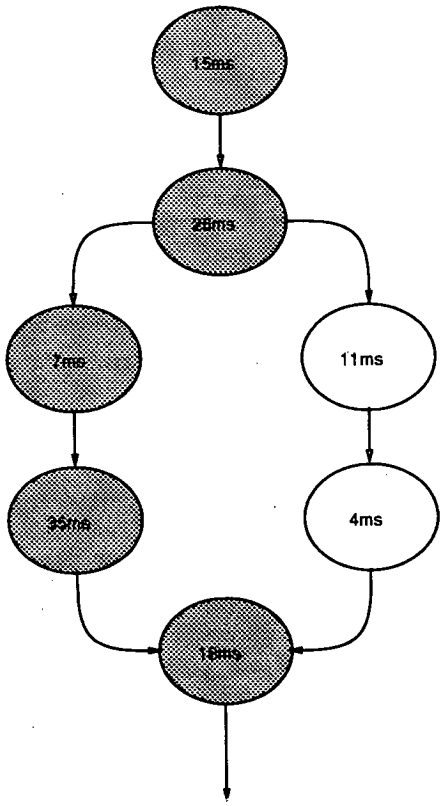


Figure 2–2: Dag with Shaded Critical Path

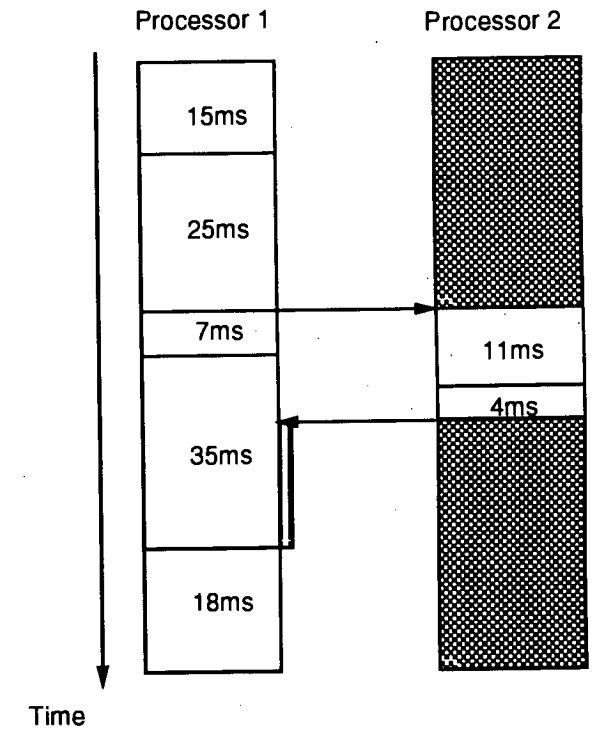


Figure 2–3: Schedule Represented as a Gantt chart

2.3 Scheduling with Comunication Delays

The model described in Section 2.2 captures the essence of interprocessor communication in terms of the implied precedence, but fails to capture any of the overheads associated with message transfer. We now introduce an approach to modelling such costs whereby messages are subject to a delay. The area has been the subject of a recent short review (Veltman *et al.* [1990]).

The three conditions of validity of the schedule are essentially the same as in Section 2.2 above, except that the time at which a task can be executed in order for the schedule not to be *temporally compromised* is determined by the time of completion of its predecessors plus the time required for the results of its predecessors to reach the processor on which it is executed. This statement is made more rigorous in Definition 2.35.

If we consider our example in Section 2.2, we can annotate each of the *edges* of the graph with a value indicating the number of bytes of data that must be communicated from the precedent task to its successor in order to satisfy the precedence relation. Such a set of edge labels is shown in Figure 2–4(a). Note that, as Efe [1982] points out, the node and edge labels are different in character: edges are labelled with data volumes and the vertices are labelled with times. In order to make the labels consistent, in the models outlined in this section there is often considered to be some *per byte* communication latency between a given pair of processors. One can think of this as the reciprocal of the available interconnect bandwidth between these processors.

We will consider mapping the computation to two processors, and we will assume a $1\mu s/byte$ communications latency. If we return to the schedule of Figure 2–3, we can respect the order of task execution on each processor but in order to allow the requisite interprocessor communication we must delay the

execution of the tasks as shown in Figure 2–4(b). As a result of this, the schedule is actually longer than the execution time that would be achieved if all tasks were executed sequentially on the same processor. A shorter time to completion can be achieved by repeating the computation of the two tasks that come first in the partial order of the dag. The resulting schedule is shown in Figure 2–4(c).

2.4 Definitions

In this section we define the majority of the notation that is used in the thesis. The notation is based on *directed* graphs and its primary purpose is to capture the various scheduling models which we analyse in Chapters 3 and 4. We develop a simpler notation based on *undirected* graphs, primarily for use in Chapters 5 and 6. In addition a few more definitions are required on a chapter-by-chapter basis, mainly in Chapters 5 and 7.

2.4.1 Cardinality, Integer Part, Absolute Value and Modulus

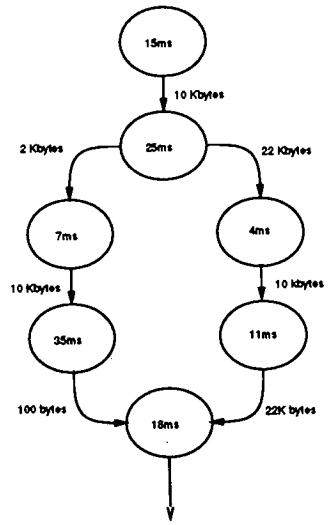
Firstly, to avoid confusion, we clarify the way we will be referring to a few simple arithmetic and set operations.

Definition 2.1 Cardinality of a set.
We refer to the *cardinality* of a set X as $|X|$.

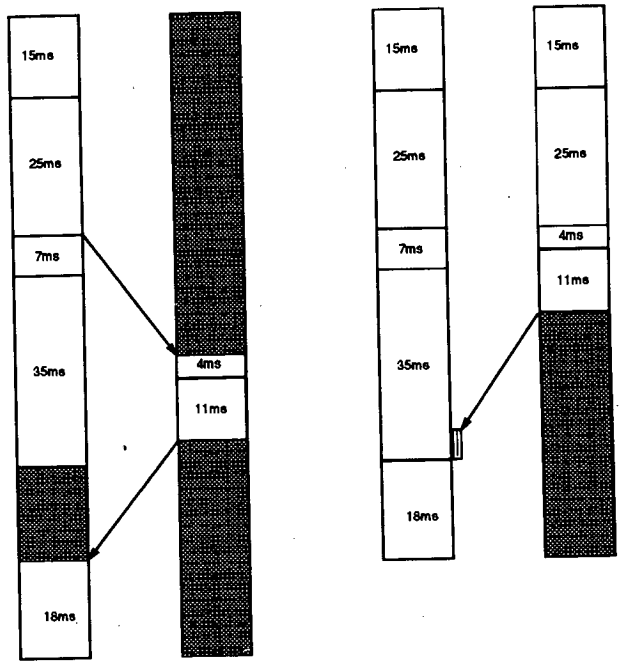
Definition 2.2 Absolute Value.
We refer to the *absolute value* of an integer X as $|X|$.

Definition 2.3 Floor.
Let R be a real number. We refer to the largest integer i such that $i \leq R$ as $\lfloor R \rfloor$

Definition 2.4 Ceiling.
Let R be a real number. We refer to the smallest integer i such that $i \geq R$ as $\lceil R \rceil$



a) Dag Annotated with Communications Volumes



b) Schedule without Recomputation

c) Schedule with Recomputation

Figure 2–4: Scheduling in a Model with Communication Delays

Definition 2.5 Integer Remainder.
Let $i \in Z_0^+$ and $j \in Z_1^+$. By $i \bmod j$ we denote $j(i/j - \lfloor i/j \rfloor)$.

2.4.2 Directed Graphs

Next we define the way we will be referring to *directed graphs* and their properties. Most of these definitions are as given by Swamy and Thulasiram [1981].

Definition 2.6 Directed Graph.
A *directed graph*, say $\Lambda = (\Gamma, \Delta)$, is an ordered pair consisting of a finite set Γ of *vertices* and a finite set Δ of *edges*. Each edge, $\rho \in \Delta$, is an ordered pair of vertices, say (γ, δ) where $\gamma, \delta \in \Gamma$.

Definition 2.7 Directed Walk.
A *directed walk* in a directed graph $\Lambda = (\Gamma, \Delta)$ is a finite sequence of vertices, say $\gamma_1, \gamma_2, \ldots, \gamma_n$, $n \geq 2$ such that for $i = 2, 3, \ldots, n$, $(\gamma_{i-1}, \gamma_i) \in \Delta$. We say that this directed walk is *from* γ_1 *to* γ_n.

Definition 2.8 Open/Closed Directed Walk.
A directed walk from γ to δ is *closed* iff, that is if and only if, $\gamma = \delta$. Otherwise it is *open*.

Definition 2.9 Directed Trail.
A directed walk, say $\gamma_1, \gamma_2, \ldots, \gamma_n$ is a *directed trail* iff each (γ_{i-1}, γ_i), $i = 2, 3, \ldots, n$, is distinct. We say that this directed trail is *from* γ_1 *to* γ_n.

Definition 2.10 Directed Path.
An open directed trail, say $\gamma_1, \gamma_2, \ldots, \gamma_n$ is a *directed path* iff each γ_i, $i = 1, \ldots, n$, is distinct. We say that this directed path is *from* γ_1 *to* γ_n.

Definition 2.11 Directed Cycle.
A closed directed trail, say $\gamma_1, \gamma_2, \ldots, \gamma_n$ is a *directed cycle* iff each γ_i, $i = 1, \ldots, n$, is distinct, except $\gamma_1 = \gamma_n$.

Definition 2.12 dag.

A directed graph is *acyclic* if it has no directed cycles. Such a graph is referred to as a dag[1].

Definition 2.13 Predecessors; Successors.

Let $\Lambda = (\Gamma, \Delta)$ be a dag. Let $\gamma \in \Gamma$. The *predecessor set*, $pred(\gamma, \Lambda)$, of γ is defined to be the set $\{\delta$: there is a directed path from δ to γ in $\Lambda\}$. The *successor set* $succ(\gamma, \Lambda)$ is defined similarly.

Definition 2.14 Antichain.

An *antichain* of a dag $\Lambda = (\Gamma, \Delta)$ is a set $\Gamma' \subseteq \Gamma$ of vertices such that for each pair of vertices $\gamma, \delta \in \Gamma'$, $\gamma \neq \delta$, there is neither a directed path from γ to δ nor a directed path from δ to γ in Λ.

Definition 2.15 Width.

The *width*, $\mathcal{W}(\Lambda)$ of a dag $\Lambda = (\Gamma, \Delta)$ is the cardinality of a maximal-size antichain in Λ.

Definition 2.16 Height.

The *height*, $\mathcal{H}(\Lambda)$ of a dag $\Lambda = (\Gamma, \Delta)$ is the cardinality of a maximal-size directed path in Λ.

2.4.3 Models

Our scheduling models consist of dags which encode tasks and the precedence relationship between them, processors which are assumed to be identical, and functions which encode the costs associated with tasks and with the communications implied by precedence edges. We start by defining the most general scheduling model that we use, and then define special cases of it.

---

[1]This is an abbreviation for *directed acyclic graph*, although *acyclic directed graph* is a better description.

Definition 2.17 General Delay Scheduling Model.
An instance of a *General Delay* scheduling model is a 4-Tuple (P, Λ, f, d) where P is a set of n processors, $\Lambda = (\Gamma, \Delta)$ is a dag such that Γ is a set of tasks, $f : \Gamma \rightarrow Z_1^+$ is a function returning the time that a task requires for execution[2], and $d : \Delta \times P \times P \rightarrow Z_0^+$ is a function where $d((\gamma, \delta), p, q)$ returns the delay associated with communication if task γ is mapped to processor p and task δ is mapped to processor q.

It is useful to classify restricted versions of the *General Delay* scheduling model.

Definition 2.18 UET Scheduling Model.
Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a *General Delay* scheduling model. A is said to be an instance of a *Unit Execution Time* (UET) model iff the range of f is $\{1\}$.

Definition 2.19 No Communication Scheduling Model.
Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a *General Delay* scheduling model. A is said to be an instance of a *No Communication* model iff $\Delta = \{\}$.

Definition 2.20 No Delay Scheduling Model.
Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a *General Delay* scheduling model. A is said to be an instance of a *No Delay* model iff the range of d is $\{0\}$.

Definition 2.21 Unit Delay Scheduling Model.
Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a *General Delay* scheduling model. A is said to be an instance of a *Unit Delay* model iff for all $p \in P$, for all $\rho \in \Delta$, $d(\rho, p, p) = 0$, and for all $p, q \in P$ where $p \neq q$, for all $\rho \in \Delta$, $d(\rho, p, q) = 1$.

---

[2]In Chapter 7 we will also have use for a *General Delay* scheduling model which allows f to return 0

Definition 2.22 Fixed Delay Scheduling Model.
Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a *General Delay* scheduling model. A is said to be an instance of a *Fixed Delay* model iff for all $p \in P$, for all $\rho \in \Delta$, $d(\rho, p, p) = 0$, and there exists some non-negative integer τ such that for all $p, q \in P$ where $p \neq q$, for all $\rho \in \Delta$, $d(\rho, p, q) = \tau$.

Definition 2.23 Uniform Delay Scheduling Model.
Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a *General Delay* scheduling model. A is said to be an instance of a *Uniform Delay* model iff for all $p \in P$ for all $\rho \in \Delta$, $d(\rho, p, p) = 0$, and for each $\rho \in \Delta$ there exists some non-negative integer η_ρ such that for all $p, q \in P$, $p \neq q$, $d(\rho, p, q) = \eta_\rho$.

Section 2.1 described a *No Communication* scheduling model. Section 2.2 described a *No Delay* scheduling model. Section 2.3 described a *General Delay* scheduling model.

It should be clear that there is a hierarchy of generality which means that *General Delay* scheduling problems are at least as hard as *Uniform Delay* scheduling problems which are at least as hard as *Fixed Delay* problems which are at least as hard as either *Unit Delay* scheduling problems or *No Delay* scheduling problems, and that *Unit Delay* scheduling problems or *No Delay* problems are at least as hard as *No Communication* scheduling problems.

2.4.4 Schedules

Given a scheduling model, the problem of allocating computation to processors becomes that of finding a schedule (that is, an allocation of tasks to processors) such that the constraints outlined in Section 2.2 are respected. A given task may be allocated to more than one processor. We define a schedule as a set of *tuples*, where a tuple refers to the execution of a particular task on a particular processor at a particular time. Note that at the point of definition, we are not interested in the validity or otherwise of schedules.

Definition 2.24 Schedule.

We define a *schedule* of a set of tasks Γ onto a set of processors P, as a finite set of 3-tuples $\{x_1, x_2, \ldots x_n\}$ where for $i = 1, 2, \ldots, n$, $x_i = (\gamma_i \in \Gamma, p_i \in P, t_i \in Z_0^+)$. The tuple x_i indicates that task γ_i is executed on processor p_i beginning at time t_i.

For notational convenience we will often wish to refer only to certain tuples within a schedule, such as those which specify a particular processor. The following notation is widely used in the thesis to describe a subset of a schedule. Note that, according to the above definition, the resulting subset is also a schedule.

Definition 2.25 Matching Subset of Schedule.

Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a *General Delay* scheduling model. Let s be a schedule for A. We use the notation $X(s, \Gamma', P', T')$ as shorthand for

$$\{(\gamma, p, t) \in s : \gamma \in \Gamma', p \in P', t \in T'\},$$

where it is understood that $\Gamma' \subseteq \Gamma$,$P' \subseteq P$ and $T' \subseteq Z_0^+$.

The following definitions are more pieces of shorthand which are used to refer to certain properties of schedules.

Definition 2.26 Makespan.

Let s be a schedule. We define its *makespan*, denoted $\mathcal{M}(s)$, as the lowest value of time beyond which no tuple is being executed in s. More formally we define

$$\mathcal{M}(s) = \max_{(\gamma,p,t)\in s} t + f(\gamma).$$

Definition 2.27 Idle Processor, Busy Processor.

In a schedule s of a task set Γ onto a processor set P, a processor $p \in P$ is said to be *idle* iff $X(s, \Gamma, \{p\}, Z_0^+) = \{\}$. Otherwise it is said to be *busy*.

Definition 2.28 Processor Usage.

The *processor usage*, denoted $\mathcal{P}(s)$, of a schedule s of an instance $(P, \Lambda =$

$(\Gamma, \Delta), f, d)$ of a scheduling model is the number of busy processors. More formally,

$$\mathcal{P}(s) = |\{p \in P : X(s, \Gamma, \{p\}, Z_0^+) \neq \{\}\}|.$$

Definition 2.29 Active Task Set.
We define the *active task set*, denoted $\mathcal{A}(s, x)$, of a schedule s of an instance $(P, \Lambda = (\Gamma, \Delta), f, d)$ of a scheduling model, at time x as

$$\mathcal{A}(s, x) = \{(\gamma, p, t) \in s : t \leq x < t + f(\gamma)\}.$$

2.4.5 Validity of Schedules

Having developed a notation in which schedules may be expressed, we are now able to define more formally the properties described in Section 2.2. The two properties of *completeness* and *overbookedness* are defined in a straightforward way. The definition of *temporal compromise* is made in terms of subsidiary definitions which at this stage appear superfluous, but which are of use in Chapters 6, 7 and 9.

Definition 2.30 (In)Complete Schedule.
Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a *General Delay* scheduling model. Let s be a schedule for A. s is said to be *complete* for A iff every task is executed. More formally this means that for all tasks $\gamma \in \Gamma$ $X(s, \{\gamma\}, P, Z_0^+) \neq \{\}$. If a schedule is not *complete* it is referred to as *incomplete*.

Definition 2.31 Overbooked Schedule.
Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a *General Delay* scheduling model. Let s be a schedule for A. s is said to be *overbooked* if any processor is executing more than one task at any one time. More formally this means that there exists a tuple $(\gamma, p, t) \in s$ such that $X(s - \{(\gamma, p, t)\}, \Gamma, \{p\}, \{t, t+1, \ldots, t+f(\gamma)-1\}) \neq \{\}$.

Definition 2.32 (Interprocessor) Tuple Dependence.

Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a general delay scheduling model, and s be a schedule for A. A *tuple dependence* for s is an ordered pair of tuples, say $c = ((\gamma, p, t), (\delta, q, t'))$ where $(\gamma, q, t'), (\delta, p, t) \in s$ and $(\gamma, \delta) \in \Delta$. c is an *interprocessor* tuple dependence iff $p \neq q$.

Definition 2.33 Valid Tuple Dependence.

Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a general delay scheduling model, and s be a schedule for A. Let $c = ((\gamma, p, t), (\delta, q, t'))$ be a tuple dependence for s. c is *valid* for s iff

$$t + f(\gamma) + d((\gamma, \delta), p, q) \leq t'.$$

Definition 2.34 (Valid) Tuple Dependence Graph.

Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a *General Delay* scheduling model, and s be a schedule for A. A *tuple dependence graph*, say $D = (s, C)$ is a directed graph with vertex set s and with a set of tuple dependences, C. D is valid for s iff both of the following are true. First, all dependences in C are valid for s and second, for each tuple in s, for each incoming edge to the task in the tuple, there is at least one tuple dependence in C. More formally, $\forall(\delta, q, t') \in s \; \forall \gamma \in \Gamma$ such that $(\gamma, \delta) \in \Delta$, $\exists((\gamma, p, t), (\delta, q, t')) \in C$ such that

$$t' \geq t + f(\gamma) + d((\gamma, \delta), p, q).$$

Definition 2.35 Temporally Compromised Schedule.

Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a *General Delay* scheduling model, and s be a schedule for A. s is said to be *temporally compromised* iff it has no valid tuple dependence graph, that is there exists a tuple $(\delta, q, t) \in s$ and an edge $(\gamma, \delta) \in \Delta$ such that

$$\min_{(\gamma, p, t') \in X(s, \{\gamma\}, P, Z_0^+)} t' + f(\gamma) + d((\gamma, \delta), p, q) > t.$$

Definition 2.36 Valid Schedule.
A schedule will be said to be *valid* iff it is *complete* and neither *overbooked* nor *temporally compromised.*

2.4.6 Properties of Schedules

Finally we define some properties (other than validity) of schedules and some ways in which schedules may be related. In later chapters we will prove further general properties about schedules which have these basic properties, or are related to each other in these ways.

Definition 2.37 Perfect Schedule.
A schedule s of an instance $(P, \Lambda = (\Gamma, \Delta), f, d)$ of a scheduling model, is referred to as a *perfect schedule* iff it is valid and for all times $x = 1, 2, \ldots, \mathcal{M}(s)$, $|\mathcal{A}(s, x)| = |P|$. Informally each processor is active throughout the whole computation.

Definition 2.38 Pruning.
A schedule s' is defined to be a *pruning* of a schedule s iff $s' \subset s$.

Definition 2.39 Fully Pruned.
A schedule s is said to be *fully pruned* with respect to an instance of a scheduling model iff, with respect to that instance, it is valid, and there exists no valid schedule $s' \subset s$.

Definition 2.40 Remapping.
A schedule s' is defined to be a *remapping* of a schedule s of an instance $(P, \Lambda = (\Gamma, \Delta), f, d)$ of a scheduling model iff it executes the same tasks as s at the same times as in s, but not necessarily on the same processors. More formally, this means $\forall(\gamma, p, t) \in s$, $\exists q \in P$ such that $(\gamma, q, t) \in s'$ and $\forall(\gamma, q, t) \in s'$, $\exists p \in P$ such that $(\gamma, p, t) \in s$.

Chapter 3

Analysis of Scheduling Models

In this chapter we review and derive a number of results for a variety of scheduling models, and put the models into the framework given in Chapter 1. The results concern the way in which performance gains can be achieved by adding processors, and the complexity of scheduling to achieve a given deadline. These results turn out to be strongly dependent upon the exact nature of the scheduling model. It is our purpose to highlight the differences between the models. As a result of the bounds derived, and assuming some basic properties of schedules, we are able to show differences between the various scheduling models in terms of our framework.

To start with, in Section 3.1 below, we prove a number of results which are of use in the later sections of this chapter and in Chapter 4. We also define a simple optimal scheduling algorithm for *No Delay* scheduling models where there are at least as many processors as tasks.

3.1 General-Purpose Results

The general purpose results are divided into two subsections: those that refer to *General Delay* scheduling models, and those that refer exclusively to *No Delay* scheduling models. Many of the results are relatively trivial, and appear only to avoid replication in later proofs.

3.1.1 Lemmas referring to General Delay Scheduling Models

Lemma 3.1 *Let* $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ *be an instance of a General Delay scheduling model, and* s *be a valid schedule for* A. *Let* s' *be a schedule formed by pruning* s, $\mathcal{M}(s') \leq \mathcal{M}(s)$.

Proof

By Definition 2.38 $s' \subset s$. Thus

$$\mathcal{M}(s) = \max_{(\gamma,p,t)\in s} t + f(\gamma) \geq \max_{(\gamma',p',t')\in s'} t' + f(\gamma') = \mathcal{M}(s').$$

□

Lemma 3.2 *Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a General Delay scheduling model, and s be a schedule for A which is not overbooked. Let s' be a schedule which is a pruning of s. s' is not overbooked.*

Proof

Let us assume as a hypothesis to be proven contradictory that there exists some tuple, say $(\gamma, p, t) \in s'$ such that $X(s' - \{(\gamma, p, t)\}, \Gamma, p, \{t, t+1, \ldots, t+f(\gamma)\}) = x \neq \{\}$. Then by Definition 2.25 $x \subseteq s'$. Now since by Definition 2.38 $s \supset s'$, $x \subset s$. This in turn would imply that $X(s - \{(\gamma, p, t)\}, \Gamma, p, \{t, t+1, \ldots, t+f(\gamma)\}) \supseteq x$, which contradicts our assumption that s is not *Overbooked*. □

Lemma 3.3 *Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a General Delay scheduling model, s be a schedule for A and s' be a schedule formed by remapping s. $\mathcal{M}(s') = \mathcal{M}(s)$.*

Proof

By Definition 2.26 there exists a tuple $(\gamma, p, t) \in s$ such that $t + f(\gamma) = \mathcal{M}(s)$. By Definition 2.40 this implies there is a tuple $(\gamma, p', t) \in s'$ and so $\mathcal{M}(s') \geq \mathcal{M}(s)$.

Now let us assume as a hypothesis to be proven contradictory that $\mathcal{M}(s') > \mathcal{M}(s)$. Thus, by Definition 2.26 there exists a tuple $(\gamma', q, t') \in s'$ such that $t + f(\gamma') = \mathcal{M}(s')$. Thus, by Definition 2.40 there is a tuple $(\gamma', q, t') \in s$, corresponding to the execution of a task which terminates at time $\mathcal{M}(s')$. By hypothesis $\mathcal{M}(s') > \mathcal{M}(s)$ which contradicts Definition 2.26. Thus we may conclude that $\mathcal{M}(s) = \mathcal{M}(s')$. □

Lemma 3.4 *Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a General Delay scheduling model, s be a complete schedule for A and s' be a schedule formed by remapping s. s' is complete.*

Proof

Let us assume as a hypothesis to be proven contradictory that s' is not complete. Thus, by Definition 2.30 there exists some task γ such that $X(s', \{\gamma\}, P, Z_0^+) = \{\}$. By Definition 2.40, $\forall(\gamma, p, t) \in s$, $\exists q \in P$ such that $(\gamma, q, t) \in s'$. Thus $X(s, \{\gamma\}, P, Z_0^+) = \{\}$. Thus s is incomplete, which leads to a contradiction. □

Lemma 3.5 *Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a General Delay scheduling model, and s be a valid schedule for A. If in Λ, $\gamma \in pred(\delta, \Lambda)$ and there exists a tuple $(\delta, p, t_\delta) \in s$ then there exists a tuple $(\gamma, q, t_\gamma) \in s$ such that $t_\gamma + f(\gamma) \leq t_\delta$.*

Proof

Let us assume as a hypothesis to be proven contradictory that $\delta \in \Gamma$ and $\gamma \in pred(\delta, \Lambda)$ and there exists some tuple $(\delta, p, t_\delta) \in s$ but no tuple $(\gamma, q, t_\gamma) \in s$ such that $t_\gamma + f(\gamma) \leq t_\delta$. Let $\gamma = \gamma_1, \gamma_2, \ldots, \gamma_x = \delta$ be a directed path in Λ.

For $i = 2, \ldots, x$, $(\gamma_{i-1}, \gamma_i) \in \Delta$ and thus, by Definition 2.35 for each tuple $(\gamma_i, p, t_{\gamma_i}) \in X(s, \{\gamma_i\}, P, Z_0^+)$, there exists a tuple, say

$$(\gamma_{i-1}, p', t_{\gamma_{i-1}}) \in X(s, \{\gamma_{i-1}\}, P, Z_0^+)$$

such that

$$t_{\gamma_{i-1}} + f(\gamma_{i-1}) + d((\gamma_{i-1}, \gamma_i), p', p) \leq t_{\gamma_i}.$$

However, by Definition 2.17, $d((\gamma_{i-1}, \gamma_i), p', p) \geq 0$, and thus $t_{\gamma_{i-1}} + f(\gamma_{i-1}) \leq t_{\gamma_i}$. In addition, by Definition 2.17, $f(\gamma_{i-1}) > 0$. Thus $t_{\gamma_{i-1}} + f(\gamma_{i-1}) < t_{\gamma_i} + f(\gamma_i)$, $i = 2, 3, \ldots, x$, that is

$$t_\gamma = t_{\gamma_1} + f(\gamma_1) \leq t_{\gamma_2} + f(\gamma_2) \ldots \leq t_{\gamma_x} = t_\delta$$

which leads to a contradiction. □

3.1.2 Results Referring Only to No Delay Scheduling Models

Our first result below states that we can remap schedules of *No Delay* scheduling models without worrying about rendering them temporally compromised.

Lemma 3.6 *Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a No Delay scheduling model. Let s be a valid schedule for A. Let s' be a remapping of s. s' is not temporally compromised*

Proof

Let us assume as a hypothesis to be proven contradictory that s' is *temporally compromised*. By Definition 2.20 $\forall p, q \in P$, $\forall \rho \in \Delta$, $d(\rho, p, q) = 0$. Thus, by Definition 2.35, there exists a tuple, say $(\gamma, p', t_\gamma) \in s'$ such that there exists an edge, say $(\delta, \gamma) \in \Delta$, such that

$$\min_{(\delta, q, t_\delta) \in X(s', \{\delta\}, P, Z_0^+)} (t_\delta + f(\delta)) > t_\gamma.$$

Now by Definition 2.40, for our tuple $(\gamma, p', t_\gamma) \in s'$ there exists a processor $p \in P$ such that $(\gamma, p, t) \in s$, and by Definition 2.36 there exist a processor $q \in P$ such that $(\delta, q, t_\delta) \in s$ and $t_\delta + f(\delta) \leq t$. This in turn means, by Definition 2.40, there must exist a a tuple $(\delta, q', t_\delta) \in s'$, such that $t_\delta + f(\delta) \leq t$. Thus we would conclude that

$$\min_{(\delta, q, t_\delta) \in X(s', \{\delta\}, P, Z_0^+)} (t_\delta + f(\delta)) \leq t_\gamma,$$

which leads to a contradiction and the lemma is proved. □

Our second result relates to the following naïve scheduling algorithm, which assigns each task γ_i to a different processor, p_i at a time which is the maximum vertex-weighted path length for those paths whose end vertex is γ_i. We prove that such schedules are valid and have optimal makespan. In due course we will define another algorithm that can be used to make such schedules less processor-greedy.

Algorithm 3.1 Naïve No Delay Scheduler

1 Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a *No Delay* Scheduling Model where $|\Gamma| = |P|$;

2 Let $p_1, \ldots, p_{|P|}$ be an enumeration of P;

3 Let $\gamma_1, \ldots, \gamma_{|P|}$ be an enumeration of γ;

4 Let $s = \{\}$;

5 For each $\gamma_i \in \Gamma$

do

6 let $t_i = \max_{\delta_1,\ldots,\delta_x,\gamma_i \text{ is a directed path in } \Gamma} \sum_{j=1}^{x} f(\delta_j)$;

7 let $s = s \cup (\gamma_i, p_i, t_i)$;

enddo

end

Theorem 3.1 *Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a No Delay Scheduling Model where $|\Gamma| = |P|$. Let s be the result of applying Algorithm 3.1 to A. s is valid and there exists no valid schedule s' such that $\mathcal{M}(s') < \mathcal{M}(s)$.*

Proof

For $i = 1, \ldots, |P|$, let p_i, γ_i and t_i be as assigned in the execution of Algorithm 3.1. Note that for each $p_i \in P$, $|X(s, \Gamma, \{p_i\}, Z_0^+)| = 1$ thus by Definition 2.31 s is not overbooked. Note also that for each task $\gamma_i \in \Gamma$ there is a tuple $(\gamma_i, p_i, t_i) \in s$, so, by Definition 2.30, s is complete.

Let us assume as a hypothesis to be contradicted that s is temporally compromised. This implies there exists a task $\gamma_a \in \Gamma$ and an edge $(\gamma_a, \gamma_b) \in \Delta$ such that there exists a pair of tuples say $(\gamma_a, p_a, t_a), (\gamma_b, p_b, t_b) \in s$ where $t_b < t_a + f(\gamma_a)$. Let $\gamma_{\Pi_1}, \ldots, \gamma_{\Pi_y}, \gamma_a, \gamma_b$ be the directed path in Γ referred to at Line 6. Thus $t_a = \sum_{j=1}^{y} f(\gamma_{\Pi_j})$. Thus $t_b < \sum_{j=1}^{y} f(\gamma_{\Pi_j}) + f(\gamma_a)$. Thus

$$t_b < \max_{\delta_1,\ldots,\delta_x,\gamma_b \text{ is a directed path in } \Gamma} \sum_{j=1}^{x} f(\delta_j)$$

which leads to a contradiction.

Let us assume as a hypothesis to be proven contradictory that there exists a valid schedule s' such that $\mathcal{M}(s') < \mathcal{M}(s)$. Let $(\gamma_a, p_a, t) \in s'$ be a tuple such that $t + f(\gamma_a) = \mathcal{M}(s')$. Let $\gamma_{\Pi_1}, \ldots, \gamma_{\Pi_y} = \gamma_a$ be a directed path in Γ such that $t_a = \sum_{j=1}^{y-1} f(\gamma_{\Pi_j})$. For each $j = 1, \ldots, y$ let t'_j be the earliest time at which t_{Π_j} is executed in s'. Note that, by Definitions 2.20 and 2.35 we have that for $j = 2, \ldots, y$, $t'_j \geq t'_{j-1} + f(j-1)$. Thus

$$t'_1 \leq t'_y - \sum_{j=1}^{y-1} f(\gamma_{\Pi_j}) \leq t - t_a < 1$$

which, by Definition 2.17 leads to a contradiction. □

3.2 Bounds on Profitable use of Processors

In this section we consider the way in which, as a result of the differences between the underlying communications costs, the choice by the multicomputer programmer of a *No Communication* model, a *No Delay* model, a *Fixed Delay* model, or a *Uniform Delay* model for performance prediction, could lead to very different conclusions as to how many processors he or she might usefully use.

3.2.1 Results for Models Without Communications Delay

Our first result simply states that in models without communication delay there is never any benefit to be gained from repeated execution of the same task.

Theorem 3.2 *Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a No Delay scheduling model. Let s be a fully pruned schedule for A. For all $\gamma \in \Gamma$, $|X(s, \{\gamma\}, P, Z_0^+)| = 1$.*

Proof

Let us assume, as a hypothesis to be contradicted, that there exists some task

γ such that $|X(s, \{\gamma\}, P, Z_0^+)| \neq 1$. We let x denote the set $X(s, \{\gamma\}, P, Z_0^+)$. By Definitions 2.31 and 2.39, $|x| \geq 1$ and thus we may assume that $|x| > 1$. Let $x_1, x_2, ..., x_n$ be an enumeration of x such that for $i = 1, 2, ..., n$, $x_i = (\gamma, p_i, t_i)$ and for $i = 2, 3, ..., n$, $t_{i-1} \geq t_i$. Let $s' = (s - x) \cup \{x_1\}$. By Definition 2.39 and by hypothesis, s' is invalid.

Since $s' \supset X(s, \Gamma - \{\gamma\}, P, Z_0^+)$, and s is valid, for all $\beta \in \Gamma - \{\gamma\}$, we have that $X(s', \{\beta\}, P, Z_0^+) \neq \{\}$. Since $s' \supset \{x_1\}$, we have that $X(s, \{\gamma\}, P, Z_0^+) \neq \{\}$. Thus we conclude that for all $\beta \in \Gamma$, $X(s', \{\beta\}, P, Z_0^+) \neq \{\}$, so by Definition 2.30, s is complete.

Let us assume s' is *temporally compromised*. Since s is valid, $s \cup s' = s$ and $s - s'$ contains only tuples instancing γ and by Definitions 2.35 and 2.20

$$t_1 + f(\gamma) > \min_{(\gamma,p,t) \in X(s,\{\gamma\},P,Z_0^+)} t + f(\gamma).$$

However, $\min_{(\gamma,p,t) \in X(s,\{\gamma\},P,Z_0^+)} t = t_1$ and so we conclude s' is not temporally compromised.

By Lemma 3.2, $s' \subset s$ cannot be overbooked.

Thus we must conclude that s' is valid since it is *complete* and neither *temporally compromised* nor *overbooked*, which leads to a contradiction, and so the theorem is proved. □

Next, as an intermediate result which is of use in proving a number of theorems, we show that in any schedule without repeated execution of the same task, at all time steps the active task set of Definition 2.29 is an antichain.

Lemma 3.7 *Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a General Delay scheduling model. Let s be a valid schedule for A. If for all $\gamma \in \Gamma$, $|X(s, \{\gamma\}, P, Z_0^+)| = 1$, then for all $t = 0, 1, \ldots, \mathcal{M}(s) - 1$ $\{\gamma : (\gamma, p, t) \in \mathcal{A}(s, t)\}$ is an antichain.*

Proof

Let us assume, as a hypothesis to be contradicted, that there exists some time

$t^* \in \{0, 1, \ldots, \mathcal{M}(s) - 1\}$ such that $\{\gamma : (\gamma, p, t^*) \in \mathcal{A}(s, t^*)\}$ is not an antichain. Thus there exists some pair of tuples, say $(\delta, q, t), (\gamma, p, r) \in \mathcal{A}(s, t^*)$, such that $\gamma \in pred(\delta, \Lambda)$. Thus $r \leq t^* < r + f(\gamma)$ and $t \leq t^* < t + f(\delta)$ in which case $t < r + f(\gamma)$.

By Lemma 3.5 in order for s to be valid there would have to be a tuple $(\gamma, p, r^*) \in s$ such that $r^* + f(\gamma) \leq t$, that is $r^* \neq r$. Thus $|X(s, \{\gamma\}, P, Z_0^+)| > 1$, which leads to a contradiction and the lemma is proved. □

This leads us on to proving that when scheduling any dag in a model in which there are no communication delays there is never any point in using more processors than the width of the dag. Consider the following algorithm which acts upon schedule s. The algorithm enumerates the set of processors in the instance of the scheduling problem and remaps all computation to a subset of that set of processors. It simply moves through the schedule considering the tuples in order of increasing time of execution. Let us say it is considering a tuple (γ, p, t) and p is outwith the chosen subset of processors, and there is some processor q within the subset which is idle at time t. Informally, the algorithm swaps the computation performed by processors p and q from timestep t onwards. The algorithm remaps all tasks executed on processor p, in timesteps t and after, so that they are executed on processor q at the same time steps, and similarly remaps q's computation to p.

Algorithm 3.2 General-Purpose Remapper

1 Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a *General Delay* Scheduling Model;

2 Let s be a valid schedule for A;

3 Let $p_1, \ldots, p_{|P|}$ be an enumeration of P;

4 Let $r_1, \ldots, r_{|s|}$ be an enumeration of s such that for each $i = 2, \ldots, |s|$, where say $r_{i-1} = (\gamma, p, t)$ and $r_i = (\delta, q, t')$, $t' \geq t$;

5 Let $x = \max_{t=0}^{\mathcal{M}(s)} |\mathcal{A}(s, t)|$;

6 For $i = 1, \ldots, |s|$

do

7 Let $(\gamma, p_j, t) = r_i$;

8 if $j > x$

do

9 let h be the minimum integer such that $X(A(s,t), \Gamma, \{p_h\}, Z_0^+) = \{\}$;

10 let $a = X(\{r_i, \ldots, r_{|s|}\}, \Gamma, \{p_j\}, Z_0^+)$;

11 let $b = X(\{r_i, \ldots, r_{|s|}\}, \Gamma, \{p_h\}, Z_0^+)$;

12 for all $r_k = (\gamma, p_j, t) \in a$ let $r_k = (\gamma, p_h, t)$;

13 for all $r_k = (\gamma, p_h, t) \in b$ let $r_k = (\gamma, p_j, t)$;

enddo

enddo

end

Note that in the above algorithm r and s are being treated like program variables. At any one instant they form a set, but as the elements of the set are modified, the set changes.

In the following lemma we prove certain properties of schedules resulting from the application of this algorithm to valid schedules of *General Delay* scheduling models. We do not go so far as to show they are valid, but merely show they are complete, not temporally compromised, and use at most as many processors as the maximum number of processors in use at any one time in the input schedule. In later theorems we will show that the output schedules are valid in three special cases. One of these cases is where the input schedule is a valid schedule for a *No Delay* scheduling model. Note that this means that application of Algorithm 3.2 to the result of applying Algorithm 3.1 to a *No Delay* scheduling model will result in a schedule with optimal makespan.

Lemma 3.8 *Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a General Delay scheduling model. Let s be a valid schedule for A. Let s' be the output of Algorithm 3.2 on s. s' is a remapping of s which is complete for A and not overbooked and such that $\mathcal{P}(s') \leq \max_{t=0}^{\mathcal{M}(s)} |\mathcal{A}(s,t)|$.*

Proof

Let $r_1, \ldots, r_{|s|}$ and $p_1, \ldots, p_{|P|}$ be the enumeration of s and p respectively chosen by the algorithm. Let $x = \max_{t=0}^{\mathcal{M}(s)} |\mathcal{A}(s,t)|$. We use variables indexed by a superscript, eg s^m, to refer to the state of the variable at Line 6 when $i = m$. Note that Lines 12 and 13 ensure that for each tuple r_a^b, $a = 1, \ldots, |s|$, $a = 1, \ldots, |s|$, where say $r_a^b = (\gamma, p, t)$, there exists some processor p' such that $r_a^{b+1} = (\gamma, p', t)$. Thus for each $i = 1, \ldots, |s|$, s^i is a *remapping* of s^{i-1}. Thus $|\mathcal{A}(s^i, t)| = |\mathcal{A}(s^{i-1}, t)| = |\mathcal{A}(s,t)|$, and, by Lemma 3.4, s' is complete.

Let us assume as a hypothesis to be proven contradictory that there exists $(\gamma, p_j, t) \in s'$ such that $j > x$. Since each tuple $r_i \in s$ is considered at Line 6 in order of increasing i and no tuple that has been considered is subsequently remapped this implies that at some value of i, where say, $(\gamma, p, t) = r_i^i$, for $m = 1, \ldots, x$, $X(\mathcal{A}(s^i, t), \Gamma, \{p_m\}, Z_0^+) \neq \{\}$ and thus $|X(\mathcal{A}(s^i, t), \Gamma, P, Z_0^+)| \geq x$. But $(\gamma, p, t) \in \mathcal{A}(s^i, t)$ and $(\gamma, p, t) \notin X(\mathcal{A}(s^i, t), \Gamma, P, Z_0^+)$, thus $|\mathcal{A}(s^i, t)| > x$ which leads to a contradiction. Thus we conclude that no tuple outside $p_1, \ldots, p_x$ is used in s' and thus $\mathcal{P}(s') \leq \max_{t=0}^{\mathcal{M}(s)} |\mathcal{A}(s,t)|$.

Let us assume as a hypothesis to be proven contradictory that s' is overbooked. Since s is not overbooked, this would imply there were two intermediate schedules, say s^i and s^{i+1} such that s^i was not overbooked and s^{i+1} was overbooked. We shall denote the value of $a^i \cup b^i$ immediately *after* Lines 12 and 13 as c. Since $s^{i+1} = (s^i - (a^i \cup b^i)) \cup c$ there must exist two tuples, say $(\gamma, p, t), (\delta, p, t') \in c$ such that $t \leq t' + f(\gamma)$. This implies there exist two tuples, say $(\gamma, p', t), (\delta, p', t') \in (a^i \cup b^i)$ such that $t \leq t' + f(\gamma)$ which implies s^i is overbooked which leads to a contradiction. □

We are now in a position to prove, by stringing together a series of lemmas, that a schedule in a *No Delay* scheduling model can usefully use no more processors than the width of the dag being scheduled.

Theorem 3.3 *Let $A = (P, \Lambda, f, d)$ be an instance of a No Delay scheduling model. Let s be a valid schedule for A. There exists a valid schedule s' such that $\mathcal{M}(s') \leq \mathcal{M}(s)$ and $\mathcal{P}(s') \leq \mathcal{W}(\Lambda)$.*

Proof
Let $s^* \subseteq s$ be a fully pruned schedule for A. Let s' be the output of Algorithm 3.2 on s^*. By Lemma 3.1 $\mathcal{M}(s^*) \leq \mathcal{M}(s)$. By Lemmas 3.3 and 3.8 $\mathcal{M}(s') = \mathcal{M}(s^*)$.

By Lemma 3.7 and Theorem 3.2, for $t = 0, 1, \ldots \mathcal{M}(s^*) - 1$, $|\mathcal{A}(s^*, t)| \leq \mathcal{W}(\Lambda)$. By Lemma 3.8 $\mathcal{P}(s') \leq \max_{t=0}^{\mathcal{M}(s)} |\mathcal{A}(s^*, t)|$ and s' is complete and not overbooked. By Lemma 3.6 s' is not temporally compromised. Thus s' is valid for A and is our schedule as required. □

Corollary 3.1 *Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a No Communication scheduling model. Let s be a valid schedule for A. There exists a valid schedule s' such that $\mathcal{M}(s') \leq \mathcal{M}(s)$ and $\mathcal{P}(s) \leq |\Gamma|$.*

For a graph with an empty edge set, Γ is an antichain, since there is no path between any of the tasks, and thus $\mathcal{W}(\Lambda) = |\Gamma|$. Note also that the following theorem establishes that the $\mathcal{W}(\Lambda)$ bound is tight.

Theorem 3.4 *Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a No Communication UET scheduling model. Let s be a valid schedule for A such that $\mathcal{M}(s) = 1$. $\mathcal{P}(s') \geq |\Gamma|$.*

Proof
Let us assume there exists a schedule valid s for A such that $\mathcal{P}(s) < |\Gamma|$. Since s is complete for some processor $p \in P$ this implies $|X(s, p, \Gamma, 0)| = 2$ which implies s is overbooked which leads to a contradiction. □

3.2.2 Precedence With Delay but Without Replication

We now consider the introduction of communications delay into the scheduling problem. Our first result refers again to Algorithm 3.2. We prove that if the input schedule to Algorithm 3.2 is a valid schedule of an instance of a scheduling model with certain properties then the output schedule is valid. These properties are either that the scheduling model is a *Unit Delay* model, or that the scheduling model is a *Fixed Delay* model, and in the schedule all tuples that are executed by a processor are executed in a single continuous period. The latter pair of properties are not important in this section, but become relevant in Section 4.2.4 below.

Lemma 3.9 *Let* $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ *be an instance of a Fixed Delay scheduling model where the range of* d *is* $\{0, \tau\}$. *If* $\tau = 1$ *let* s *be an arbitrary valid schedule for* A. *If* $\tau > 1$ *let* s *be a schedule such that there does not exist a pair of tuples* $(\gamma, p, t), (\delta, p, t')$ *such that there exists a time* t^*, $t \leq t^* \leq t'$ *such that* $X(\mathcal{A}(s, t^*), \Gamma, \{p\}, Z_0^+) = \{\}$. *Let* s' *be the result of applying Algorithm 3.2 to* s. s' *is not temporally compromised for* A.

Proof

Let r_a^b, $a = 1, \ldots, |s|, b = 1, \ldots, |s|$ refer to the state of r_a at step 6 of Algorithm 3.2 when $i = b$. Let $r_a^{|s|+1}$, refer to the state of r_a at the termination of the algorithm. Let us assume s' is temporally compromised. This implies that there exists an edge $(\gamma, \delta) \in \Delta$ and a pair of tuples, say $r_u^{|s|+1} = (\gamma, p_l, t)$ and $r_v^{|s|+1} = (\delta, p_m, t')$ such that

$$t + f(\gamma) + d((\gamma, \delta), p_l, p_m) > t'.$$

Now s is valid, and thus not temporally compromised and so there exist two tuples $r_u^1 = (\gamma, p, t)$ and $r_v^1 = (\delta, p', t')$ such that

$$t + f(\gamma) + d((\gamma, \delta), p, p') \leq t'.$$

These inequalities imply $d((\gamma,\delta),p,p') < d((\gamma,\delta),p_l,p_m)$, that is $d((\gamma,\delta),p,p') = 0$ and $d((\gamma,\delta),p_l,p_m) = \tau$. Thus $p = p'$ and $p_l \neq p_m$. They also imply $t' > t$ which in turn implies r_v comes later in the enumeration than r_u, that is $u < v$.

Let us consider the states of the variables at the beginning of the loop when $i = u$ and when $i = v$. Note that $r_u^u = (\gamma, p_l, t)$ and thus $r_v^u = (\delta, p_l, t')$. Note that at some value of i, $u \leq i < v$, $r_v^i = (\delta, p_l, t')$ and $r_v^{i+1} = (\delta, p_m, t')$. We have two cases whereby this could have occurred.

- $l > x$ at Line 8 in which case since $(\gamma, p_l, t) \in s'$, and we are led to a contradiction of Lemma 3.8.

- There exists a time t^*, $t \leq t^* \leq t'$ such that $X(\mathcal{A}(s_u, t^*), \Gamma, \{p_l\}, Z_0^+) = \{\}$ at Line 9. Thus, by our restrictions on s and A, $\tau = 1$ and $t + f(\gamma) + 1 > t'$. Now for $\hat{t} = t, \ldots, t' - 1$, $X(\mathcal{A}(s_u, \hat{t}), \Gamma, \{p_l\}, Z_0^+) = \{(\gamma, p_l, t)\}$, and thus $t' \geq t + f(\gamma) + 1$ which leads to a contradiction.

Thus we conclude that our hypothesis is contradictory and the theorem is proved. □

Theorem 3.5 *Let $A = (P, \Lambda, f, d)$ be an instance of a Unit Delay scheduling model. Let s be a valid schedule for A such that for all $\gamma \in \Gamma$, $|X(s, \{\gamma\}, P, Z_0^+)| = 1$. There exists a valid schedule s' such that $\mathcal{M}(s') \leq \mathcal{M}(s)$ and $\mathcal{P}(s') \leq \mathcal{W}(\Lambda)$.*

Proof

Let s' be the output of Algorithm 3.2 on s. By Lemmas 3.3 and 3.8 $\mathcal{M}(s') = \mathcal{M}(s)$.

By Lemma 3.7, for $t = 0, 1, \ldots, \mathcal{M}(s) - 1$, $|\mathcal{A}(s', t)| \leq \mathcal{W}(\Lambda)$. By Lemma 3.8 $\mathcal{P}(s') \leq \max_{t=0}^{\mathcal{M}(s)} |\mathcal{A}(s', t)|$ and s' is complete and not overbooked. By Lemma 3.9 s' is not temporally compromised. Thus s' is valid for A and is our schedule as required. □

3.2.3 Precedence with Delay and Replication

The proof of Theorem 3.5 relies on the fact that task replication is disallowed. Consider the dag shown in Figure 3–1 in which each of the tasks has unit execution time and there is a unit communication delay.

In order for the makespan of a schedule of the dag to be 4, tasks A, B, C, D must be executed on the one processor, tasks A, B, C, E on another and F, H, J, K on a third, since these chains are of length 4 and so there is no way that an interprocessor communication delay can be introduced into them. We are then required to compute task G at time 1 so that it can be ready as input to task E at time 3, and this must be done on a fourth processor as the three processors executing the chains are each busy for the duration of the computation. Thus a schedule with optimal makespan requires to use at least 4 processors, whereas the width of the dag is 3 by inspection.

We now consider arbitrary execution time and *Uniform Delay* scheduling models. Recall that, by Definition 2.23, *Uniform Delay* models are those where the communication delay is dependent upon the precedence edge but not upon the processors to which the tasks incident to the edge have been mapped.

We shall need the following lemma which states that in such models the last task executed by a processor can be pruned if it is computed by some other processor at the same or at an earlier time.

Lemma 3.10 *Let* $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ *be an instance of a Uniform Delay scheduling model. Let* s *be a valid schedule for* A. *Let* $(\gamma, p, t) \in s$ *be a tuple such that* $X(s, \Gamma, p, \{t+1, \ldots, \mathcal{M}(s)-1\}) = \{\}$ *and* $X(s, \{\gamma\}, P - \{p\}, \{0, 1, \ldots, t\}) \neq \{\}$. *Let* $s' = s - \{(\gamma, p, t)\}$. s' *is valid for* A.

Proof

Let us assume as a hypothesis to be proved contradictory that s' is *incomplete.*

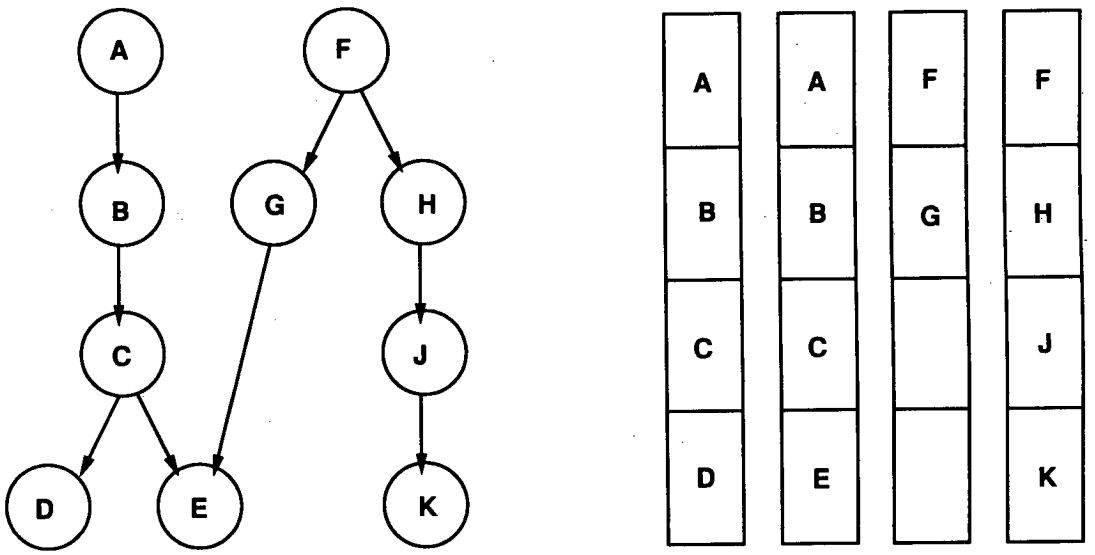


Figure 3–1: Counter-example for $\mathcal{W}(\Lambda)$ Bound in UET *Unit Delay* Scheduling models

Now s is complete thus there exists a task $\delta \in \Gamma$ such that $X(s', \{\delta\}, P, Z_0^+) = \{\}$ whereas $X(s, \{\delta\}, P, Z_0^+) \neq \{\}$. We have two cases:

- $\delta = \gamma$ in which case $X(s, \{\gamma\}, P - \{p\}, \{0, 1, \ldots, t\}) = \{\}$ which leads to a contradiction.
- $\delta \neq \gamma$ in which case since $s' = s - \{(\gamma, p, t)\}$, $X(s', \{\delta\}, P, Z_0^+) \neq \{\}$ which leads to a contradiction.

Thus we conclude that s' is complete.

Since s is valid, it is not *temporally compromised*. Let us assume as a hypothesis to be contradicted that s' is *temporally compromised*. Thus there exists some edge say $\rho = (\alpha, \beta) \in \Delta$ such that there exists a tuple $(\beta, q, t') \in s'$ which has a valid tuple dependence edge, say $((\alpha, q, t^*), (\beta, q', t'))$ in s but no such edge in s'. Since $s - s' = \{(\gamma, p, t)\}$, this implies that $(\alpha, q, t^*) = (\gamma, p, t)$. Furthermore, by Definition 2.34 $(t + f(\gamma) + d(\rho, p, q)) \leq t'$, whereas $\min_{(\gamma,p,t^*) \in s'}(t^* + f(\gamma) + d(\rho, p, q)) > t'$. That is

$$(t + f(\gamma) + d(\rho, p, q)) < \min_{(\gamma,p,t^*) \in s'}(t^* + f(\gamma) + d(\rho, p, q)).$$

Note that by Definition 2.23, there exists some non-negative integer, say η such that for all $a, b \in P, a \neq b$, $d(\rho, a, b) = \eta$. Note also that by Definition 2.17 $f(\gamma) > 0$. Since $X(s, \Gamma, p, \{t + 1, \ldots, \mathcal{M}(s) - 1\}) = \{\}$, $q \neq p$ which implies that $d(\rho, p, q) = \eta$. This in turn implies

$$t < \min_{(\gamma,p,t^*) \in s'} t^*$$

but since $X(s', \{\gamma\}, P - \{p\}, \{0, 1, \ldots, t\}) \neq \{\}$, $\min_{(\gamma,p,t^*) \in s'} t^* \leq t$, which leads to a contradiction.

By Lemma 3.2 pruning cannot make a schedule *overbooked*. Now, $s' \subset s$ that is it has been derived by pruning the valid (ie. not *overbooked*) schedule s, so we can conclude that s' is not *overbooked*.

Since s' is *complete* and neither *temporally compromised* nor *overbooked,* it is valid. □

The above lemma leads us to consider a pruning technique whereby the last tuple executed by a processor is pruned if there is another tuple instancing the same task which is executed on some other processor at a the same time or later.

Lemma 3.11 *Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a Uniform Delay scheduling model. Let s be a valid schedule for A with $\mathcal{P}(s) > |\Gamma|$. s is not fully pruned.*

Proof

Let s be a valid schedule such that $\mathcal{P}(s) > l = |\Gamma|$. For each task γ we define a sequence of tuples $\{x_1^\gamma, x_2^\gamma, \ldots, x_{n_\gamma}^\gamma\}$ such that for $i = 1, 2, \ldots, n_\gamma$, $x_i^\gamma = (\gamma, p_i^\gamma, t_i^\gamma) \in s$ and $X(s, \Gamma, p_i^\gamma, \{t_i^\gamma + 1, t_i^\gamma + 2 \ldots\}) = \{\}$ and for $i = 2, \ldots, n_\gamma$, $t_{i-1}^\gamma \leq t_i^\gamma$. Now since there are only l distinct tasks and $\mathcal{P}(s) > l$ distinct non-idle processors, by the pigeon-hole principle there must exist some task, say γ such that $n_\gamma > 1$. We then construct a schedule s' by pruning s' of x_i^γ for each $2 \leq i \leq n_\gamma$.

By Lemma 3.10 s' is valid for any instance of a *Uniform Delay* scheduling model because at each pruning of x_i $X(s, \{\gamma\}, P - \{p_i\}, \{0, 1, \ldots, t\}) \supset \{x_1\} \neq \{\}$. □

The following corollary follows from the fact that s can be iteratively pruned until $\mathcal{P}(s) \leq l$ at which point the pigeon-hole principle no longer applies.

Corollary 3.2 *Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a Uniform Delay scheduling model. Let s be a valid schedule for A with $\mathcal{P}(s) > |\Gamma|$. There exists a valid schedule s' of A such that $\mathcal{P}(s') \leq |\Gamma|$.*

Thus we have that in *Uniform Delay* models no performance gains can be achieved by using more than $|\Gamma|$ processors. Finally we show that this bound in turn relies on the *Uniform Delay* property, and that it is possible usefully to

schedule to more processors than there are tasks in the dag once this restriction is relaxed.

Theorem 3.6 *There exists an instance* $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ *of a General Delay scheduling model and an integer* M *such that for any valid schedule, say* s *of* A*, with* $\mathcal{M}(s) \leq M$*, we have that* $\mathcal{P}(s) > |\Gamma|$.

Proof

Let $\Gamma = \{\alpha\} \cup \{\beta_i : i = 1, \ldots, 3\} \cup \{\gamma_i : i = 1, \ldots, 3\} \cup \{\delta_i : i = 1, \ldots, 4\}$ be a set of tasks. Let $\Delta = \{(\alpha, \beta_1)\} \cup \{(\beta_i, \gamma_i) : i = 1, \ldots, 3\} \cup \{(\gamma_i, \delta_j) : i = 1, \ldots, 3, j = 1, \ldots, 4\}$.

Let $\Lambda = (\Gamma, \Delta)$. Λ is shown in Figure 3–2. Let $P = \{p_i^j : i = 1, \ldots, 4, j = 1, \ldots, 3\}$ be a set of processors. For all $\rho \in \Delta$ for all $p_i^j \in P$ let $d(\rho, p_i^j, p_i^j) = 0$. For all $\rho \in \Delta$ for all $p_i^j \in P$ for all $p_i^l \in P$ $j \neq l$, let $d(\rho, p_i^j, p_i^l) = 1$. For all $\rho \in \Delta$ for all $p_i^j \in P$ for all $p_k^l \in P$ $k \neq i$, let $d(\rho, p_i^j, p_k^l) = 2$. For all $\zeta \in \Gamma$ let $f(\zeta) = 1$. Let $M = 4$. Let $C_i = (c_1^i, \ldots, c_4^i) = (\alpha, \beta_1, \gamma_1, \delta_i), i = 1, \ldots, 4$. Note that each $C_1, \ldots, C_4$ is a critical path of Γ, thus s is valid only if for each $i = 1, \ldots, 4$, for each $j = 1, \ldots, 4$ there exists a tuple $(c_j^i, p_{x_i}^{y_i}, j) \in s$. Let us assume as a hypothesis to be proved contradictory that there exists some tuple $(\delta_i, p_{x_i}^{y_i}, 3) \in s$ such that for some task $\gamma \in \{\gamma_1, \gamma_2\}$,

$$X(s, \{\gamma\}, \{p_{x_i}^y : y \in \{1, \ldots, 3\} - \{y_i\}\}, \{1\}) = \{\}.$$

Now since $|pred(\gamma, \Lambda)| = 1$, it follows that $\min_{(\gamma,p,t) \in X(s,\{\gamma\},P,Z_0^+)} t = 1$ thus

$$\min_{(\gamma,p,t) \in X(s,\{\gamma\},P,Z_0^+)} t + f(\gamma) + d((\gamma, \delta_i^j), p_{x_i}^{y_i}) > 3.$$

which implies s is temporally compromised, which leads to a contradiction.

Thus we conclude that s is valid only if for each $\delta_i, i = 1, \ldots, 4$ for all processors $p_i^j, j = 1, \ldots, 3$, $X(s, \Gamma, \{p_i^j\}, 2) \neq \{\}$. This in turn implies that s is valid only if all $|P|$ processors are active in s at time 2, and since $|P| = 12 > 11 = |\Gamma|$, the theorem is proved. □

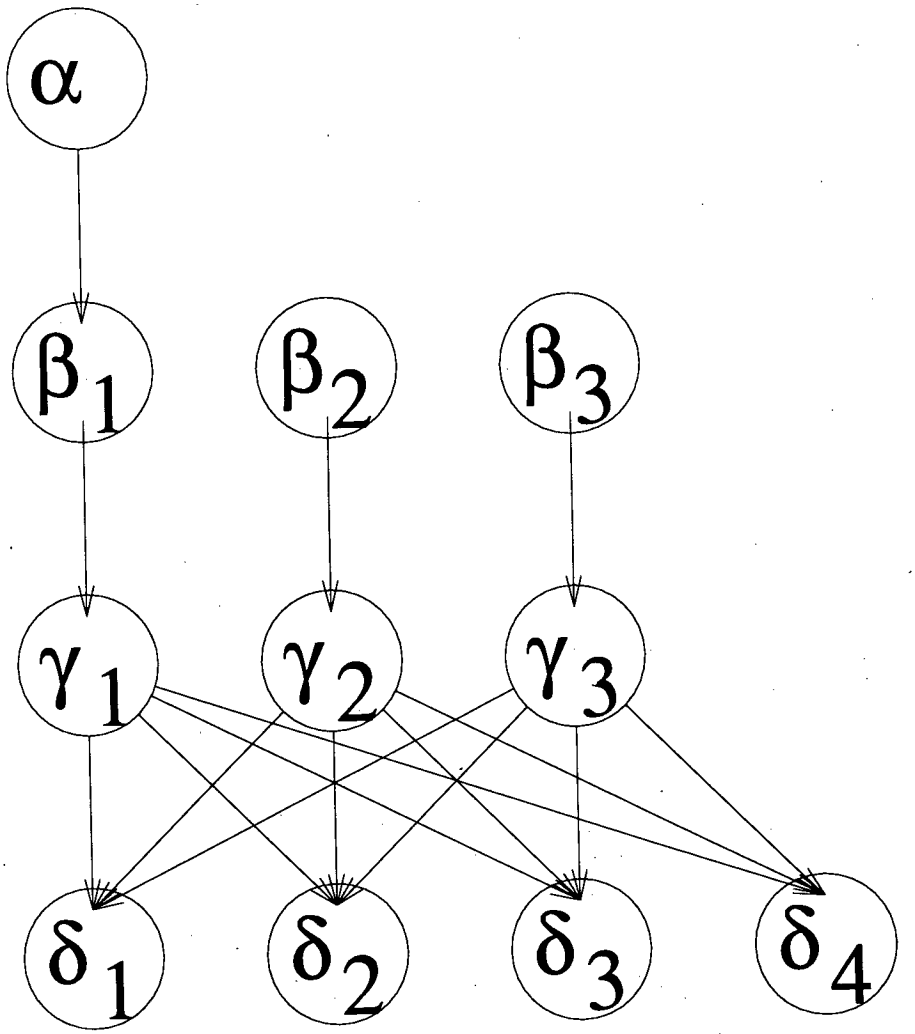


Figure 3–2: Counter-example for $|\Gamma|$ Bound in UET *General Delay* Scheduling

3.3 Complexity Results

In this section we briefly review the complexity of scheduling in the various delay-variants of scheduling models. In addition we prove a complexity result of our own.

3.3.1 The Types of Complexity Results

We can identify three different types of complexity results, namely those where the number of processors is unbounded, those where it is part of the instance and those where it is part of the problem.

The unbounded processor decision problem is stated below.

Decision Problem 3.1 *Given an instance $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ of a scheduling model where P is an infinite set, and a positive integer k does there exist a valid schedule s for A such that $\mathcal{M}(s) < k$?*

As we showed in Section 3.2 in many types of scheduling model there is a bound on the number of processors beyond which no performance gains can be achieved. In this case we can replace the infinite set of processors in Decision Problem 3.1 with a finite set of the corresponding cardinality.

The second type of decision problem is more straightforward, and can be stated as follows.

Decision Problem 3.2 *Given an instance $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ of a scheduling model and a positive integer k does there exist a valid schedule s for A such that $\mathcal{M}(s) < k$?*

The final types of scheduling problem are known as 2-processor scheduling problems, 3-processor scheduling problems etc. The x-processor scheduling problem is stated below.

Decision Problem 3.3 *Given an instance* $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ *of a scheduling model where* $|P| = x$ *and a positive integer* k *does there exist a valid schedule* s *for* A *such that* $\mathcal{M}(s) < k$*?*

It should be clear that for any particular delay-variant of the scheduling model, the NP hardness of Decision Problem 3.1 and Decision Problem 3.2 follow from the NP completeness of Decision Problem 3.3.

3.3.2 No-Communication Scheduling Models

If A is restricted to be a *No Communication* scheduling model, Decision Problem 3.3 is NP complete (Bruno *et al.* [1974]) in the case of two or more processors. It is related to the bin packing problem (See Coffman *et al.* [1984b] and Garey and Johnson [1982]).

Decision Problem 3.1 is polynomial. Indeed in these models Algorithm 3.1 above is a polynomial time algorithm for optimally scheduling to as many processors as there are tasks. Algorithm 3.2 is a polynomial time algorithm which will remap such schedules to $|\Gamma|$ processors.

There is also a set of results for preemptive scheduling, which we mention in passing. McNaughton [1959] produced a simple exact polynomial time algorithm and Martel [1988] shows that this version of the problem is in the complexity class NC defined by Pippenger [1979], which implies that it can be solved on a concurrent read concurrent write (CRCW) PRAM with a number of processors polynomial in the size of the problem in time which is a polylog of the problem size. On the other hand Rayward-Smith [1987a] shows that if preemption incurs a unit delay (in a way analogous to the communication delays

that are present in the *General Delay* scheduling model) the problem becomes NP complete.

3.3.3 No-Delay Scheduling Models

The case where A is restricted to be a *No Delay* scheduling model has been the subject of considerable analysis.

If A is restricted to be a *No Delay* UET scheduling model, Decision Problem 3.2 is NP complete. Decision Problem 3.3 has a polynomial time algorithm for $x = 2$ [Coffman and Graham 1972], and indeed is in NC [Hembold and Mayr 1987], but it is NP complete in the case of $x = 2$ if the range of f is $\{1, 2\}$ [Ullman 1975]. However, if x has a value of 3 or more (as is often the case in a real multicomputer), and A is an instance of a UET scheduling model, it is not known whether the problem is NP complete [Garey and Johnson 1979].

In these models, Decision Problem 3.1 is polynomial. Again we may use Algorithm 3.1 above to optimally schedule to as many processors as there are tasks. Algorithm 3.2 is a polynomial time algorithm which will remap such a schedule to $\mathcal{W}(\Lambda)$ processors.

3.3.4 Unit, Uniform and General Delay Models

Complexity results in this area are even now appearing in the literature. At first sight these results may seem redundant or contradictory, but it is important to take the following into consideration.

- If there is a makespan minimisation problem that is NP complete when task replication is allowed, the corresponding problem where no task replication is allowed is also NP complete. The reverse is not necessarily true.

- The NP completeness of a *No Delay* makespan minimisation problem implies the NP completeness of an equivalent problem in a *Uniform Delay*, *Fixed Delay* or *General Delay* scheduling model, but does not imply the NP completeness of the corresponding problem in a *Unit Delay* scheduling model.

- Decision Problem 3.1 is not trivial in *General Delay*, *Uniform Delay* or *Fixed Delay* scheduling models.

By a reduction which disallows recomputation, Rayward-Smith [1987b] shows that the UET *Unit Delay* scheduling problem is NP complete in the version of Decision Problem 3.2. Although he does not state the fact explicitly his encoding produces a *perfect* schedule, and thus shows the NP completeness of Decision Problem 3.2 even if replication is allowed. Decision Problem 3.3 in UET scheduling models is open for *Unit Delay*, *Fixed Delay*, *Uniform Delay* and *General Delay* models, for any $x > 1$. It is perhaps surprising that the polynomial time algorithm for $x = 2$ in *No Delay* models has not been generalised to *Unit Delay* models.

Papadimitriou and Yannakakis [1990] show that Decision Problem 3.1 is NP Complete for UET *Fixed Delay* models. Inspired by an earlier version of Papadimitriou and Yannakakis' paper, Jung *et al.* [1989] noted that, in Papadimitriou and Yannakakis's complexity proof, the encoding could produce large values for τ. Jung *et al.* present an algorithm for UET *Fixed Delay* models that finds an optimal schedule onto $|\Gamma|$ processors and proved that its time complexity is polynomial in τ. The time complexity of the algorithm is $O(l^{\tau+1})$. A similar result by Chrétienne [1989] shows a polynomial time algorithm for unbounded numbers of processors where the execution time of all tasks is less than τ.

The result of Jung *et al.* [1989] implies that, for example, UET *Unit Delay* scheduling to an unbounded number of processors has a polynomial time solution if task replication is allowed. On the other hand, Picouleau [1992] shows

that UET *Unit Delay* scheduling to an unbounded number of processors is NP complete if task replication is *not* allowed.

In the case of non UET *General Delay* models, Decision Problem 3.2 is NP complete because the *No Delay* problem is NP complete [Ullman 1975] even for 2 processors with execution time of 1 or 2. Sarkar [1989] showed Decision Problem 3.1 is NP complete for non-UET *General Delay* scheduling models. Papadimitriou and Yannakakis's later result for *Uniform Delay* models is stronger because it is for a model which is both *Uniform Delay* and UET.

Below, by a modification of Ullman's proof we show the NP completeness of *Unit Delay* two processor scheduling with execution times of 1 or 2. In our notation, as a partial result Ullman [1975] shows the NP completeness of the following problem:

Decision Problem 3.4 *Let n be a positive integer. Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a UET No Delay scheduling model where $|P| = k$ and $|\Gamma| = kn$. Does there exist a perfect No-Recomputation Schedule[1] s for A such that $\mathcal{M}(s) = n$?*

We consider the following problem:

Decision Problem 3.5 *Let n' be a positive integer. Let $B = (\{p_1, p_2\}, \Lambda', f', d)$ be an instance of a Fixed Delay scheduling model where the range of f' is $\{1, 2\}$. Does there exist a perfect No-Recomputation Schedule s' for B such that $\mathcal{M}(s') = n'$?*

Theorem 3.7 *Decision Problem 3.5 is NP complete.*

Proof

Suppose we are given an instance of Decision Problem 3.4 as above. We construct an instance of Decision Problem 3.5 as follows.

---

[1]In this case, the no-recomputation constraint is not important.

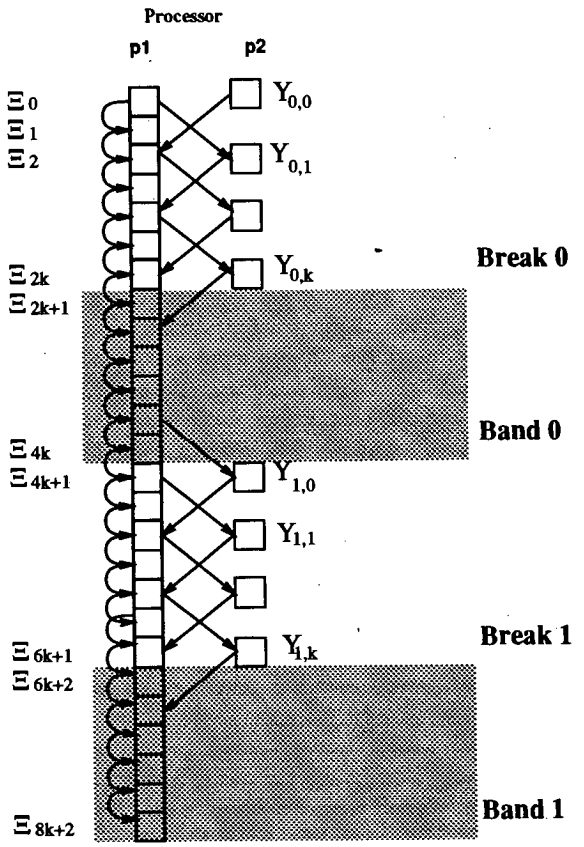


Figure 3–3: Partial Gantt Chart for the Encoding in Decision Problem 3.5

- $n' = (4k+1)n$
- $\Lambda' = (\Gamma', \Delta')$ where
- $\Gamma' = \Xi \cup \Upsilon \cup \Gamma \cup \Gamma^*$ where
  - $\Xi = \{\Xi_i : i = 0, \ldots, n'-1\}$
  - $\Upsilon = \{\Upsilon_{i,j} : i = 0, \ldots, n-1, j = 0, \ldots, k\}$
  - $\Gamma^* = \{\gamma_i^* : i = 0, \ldots, nk-1\}$
- f' is given by $f'(\alpha) = 2$ for all $\alpha \in \Gamma$ and $f'(\beta) = 1$ for all tasks $\beta \in \Gamma' - \Gamma$.
- $\Delta' = \Delta'_1 \cup \Delta'_2 \cup \Delta'_3 \cup \Delta'_4$ where
  - $\Delta'_1 = \{(\Xi_i, \Xi_{i+1}) : i = 0, \ldots, n'-2\}$,
  - $\Delta'_2 = \{(\Xi_u, \Upsilon_{i,j}), (\Upsilon_{i,j}, \Xi_{u+4}) : i = 0, \ldots, n-1,\ j = 0, \ldots, k,\ u = (4k+1)i + 2j - 2\}$ except where $u = -2$, we have only the edge $(\Upsilon_{0,0}, \Xi_2)$.
  - $\Delta'_3 = \{(\gamma_i^*, \gamma_i) | 0 \leq i < nk\}$,
  - $\Delta'_4 = \{(\gamma_i^*, \gamma_j) | (\gamma_i, \gamma_j) \in \Delta\}$.

First observe that $\sum_{\gamma \in \Gamma'} f'(\gamma) = (4k+1)2n$, and since our deadline is $(4k+1)n$ and we have two processors, any valid schedule for B is a *perfect schedule* with a single tuple instancing each task in Γ'. Next observe that because of the edges Δ'_1 one processor must be devoted to processing an element of Ξ at each time unit if the time limit is to be met, thus the tasks of Ξ in the order indicated by the precedence relation Δ'_1 form the *critical path* of the computation. Moreover, the same processor must process all elements of Ξ since no communication latency can be allowed to occur between the execution of the elements of the critical path if the computation is to finish by the deadline. Without loss of generality let us assume that it is processor p_1 that is computing the tasks in Ξ,

and therefore we have that in any valid schedule of our instance of Decision Problem 3.5, for all $\Xi_i \in \Xi$, $X(s, \{\Xi_i\}, P, Z_0^+) = \{(\Xi_i, p_1, i)\}$, and for all $\gamma \notin \Xi$, $X(s, \{\gamma\}, P, Z_0^+) = \{(\gamma, p_2, t_\gamma)\}$.

Next, the proof then follows in a very similar way to the proof described by Ullman [Ullman, 1975], pages 391–392. The tasks in Υ must be executed at very specific times as shown in Figure 3–3. Progressing in time we have an alternation of *breaks* in which there is one time unit available on processor 2 every other time unit and *bands* in which $2k$ consecutive time units are available on processor 2. Since the elements of Γ require two time units each, it is clear that they must be executed in the bands only. As there are kn such jobs, they must completely fill the bands, which means that the elements of Γ^* must be executed exclusively during the breaks.

As a consequence, if tasks $\gamma, \delta \in \Gamma$ are both executed in the same band it is not possible for there to exist an edge (γ, δ) in Δ. For if so, by our construction there would exist a task $\alpha \in \Gamma^*$ such that there exist edges (γ, α) and (α, δ) in Δ', and α would have to be executed in that same band, violating what we have just concluded. Thus if our instance of Decision Problem 3.5 has a solution, we can find a solution to the original instance of Decision Problem 3.4 by executing at time unit i exactly those jobs executed in the ith band.

Conversely if we have a solution to the given instance of Decision Problem 3.4 we can find a solution of the constructed instance of Decision Problem 3.5 by executing γ_i^* in break t and γ_i in band t whenever γ_i is executed at time unit t in the solution to Decision Problem 3.4. □

Note that the above proof establishes that the problem remains NP complete even if recomputation is allowed since s' is a *perfect schedule* without recomputation and thus if any valid schedule s^* included recomputation, $\mathcal{M}(s^*)$ would exceed n'.

3.4 Comparison with the Framework

For any valid schedule s of a *General Delay* scheduling model we can define the times that processors spend in each of the states referred to in the framework of Chapter 1.3. It is instructive to compare the values for the *No Communication, No Delay* and *General Delay* scheduling models. For the purposes of this section we assume the schedule is of a particular form: it is fully pruned and fully squashed according to the following definition.

Definition 3.1 Fully Squashed Schedule.
Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a *General Delay* scheduling model. Let s be a valid schedule for A. s is *fully squashed* iff $\forall(\gamma, p, t) \in s$

$$s \cup \{(\gamma, p, t-1)\} - \{(\gamma, p, t)\}$$

is not a valid schedule for A.

The following theorem simply states that in *No Communication* models no processor need be idle until it has finished all computation that has been assigned to it.

Theorem 3.8 *Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a No Communication scheduling model. Let s be a valid fully squashed schedule for A. For each $p \in P$,*

$$\sum_{(\gamma,p,t)\in X(s,\Gamma,\{p\},Z_0^+)} f(\gamma) = \mathcal{M}(X(s, \Gamma, \{p\}, Z_0^+)).$$

Proof
Let us assume as a hypothesis to be proven contradictory that there exists a processor $p \in P$ such that $\sum_{(\gamma,p,t)\in X(s,\Gamma,\{p\},Z_0^+)} f(\gamma) \neq \mathcal{M}(X(s, \gamma, \{p\}, Z_0^+))$. We have two cases:

- $\sum_{(\gamma,p,t)\in X(s,\Gamma,\{p\},Z_0^+)} f(\gamma) > \mathcal{M}(X(s, \Gamma, \{p\}, Z_0^+))$ in which case, by the pigeonhole principle, s is overbooked which leads to a contradiction.

- $\sum_{(\gamma,p,t)\in X(s,\Gamma,\{p\},Z_0^+)} f(\gamma) < \mathcal{M}(X(s,\Gamma,\{p\},Z_0^+))$ in which case there exists some tuple (γ,p,t) such that there exists no tuple (δ,p,t') such that $t - 1 \leq t' + f(\delta) < t$, in which case $s' = s - \{(\gamma,p,t)\} \cup \{(\gamma,p,t-1)\}$ is not overbooked. Note s' is complete and since Λ is empty s' cannot be temporally compromised. Thus we conclude that s' is valid which leads to a contradiction.

□

Let $\Lambda = (\Gamma,\Delta)$ be a dag. Let $\Lambda_A = (\Gamma,\{\})$. Let $A = (P,\Lambda_A,f,d_A)$ be an instance of a *No Communication* scheduling model. Let $B = (P,\Lambda,f,d_B)$ be an instance of a *No Delay* scheduling model. Let $C = (P,\Lambda,f,d_C)$ be an instance of a *Constant Delay* scheduling model where the range of d_C is $\{0,\tau\}$.

Let a be a fully pruned fully squashed valid schedule for A. Let b be a fully pruned fully squashed valid schedule for B. Let c be a fully pruned fully squashed valid schedule for C. Note that by Theorem 3.2, for all $\gamma \in \Gamma$ $|X(a,\{\gamma\},P,Z_0^+)| = |X(b,\{\gamma\},P,Z_0^+)| = 1$. Note also that by Theorem 3.8 $\sum_{(\gamma,p,t)\in X(a,\Gamma,\{p\},Z_0^+)} f(\gamma) = \mathcal{M}(X(a,\Gamma,\{p\},Z_0^+))$. In the following tables, p refers to some arbitrary processor $p \in P$ which is busy in the relevant schedule. Table 3–3, requires one more definition. For each $p \in P$, let $(\gamma_p^1,p,t_p^1),\ldots,(\gamma_p^{x_p},p,t_p^{x_p})$ be the enumeration of $X(s,\Gamma,\{p\},Z_0^+)$ such that for each $j = 2,\ldots,x_p$, $t_p^{j-1} + f(\gamma_p^{j-1}) \leq t_p^j$. Note that in Table 3–3 we are unable to apportion T_{calc} to an individual processor.

$T_{Calc}(p)$		$\mathcal{M}(X(a,\Gamma,\{p\},Z_0^+))$
T_{Comm}		0
T_{House}		0
T_{Idle}	$T_{Wait}(p)$	0
	$T_{Fin}(p)$	$\mathcal{M}(a) - \mathcal{M}(X(a,\Gamma,\{p\},Z_0^+))$

Table 3–1: A *No Communication* Scheduling Model in Terms of our Framework

$T_{Calc}(p)$		$\sum_{(\gamma,p,t)\in X(b,\Gamma,\{p\},Z_0^+)} f(\gamma)$
T_{Comm}		0
T_{House}		0
T_{Idle}	$T_{Wait_D}(p)$	0
	$T_{Wait_S}(p)$	$\mathcal{M}(X(b,\Gamma,\{p\},Z_0^+)) - T_{calc}(p)$
	$T_{Fin}(p)$	$\mathcal{M}(b) - \mathcal{M}(X(b,\Gamma,\{p\},Z_0^+))$

Table 3–2: A *No Delay* Scheduling Model in Terms of our Framework

T_{Calc}		$\sum_{\gamma\in\Gamma} f(\gamma)$
T_{Comm}		0
T_{House}	T_{Recalc}	$\sum_{(\gamma,p,t)\in s} f(\gamma) - T_{Calc}$
	T_{Inter}	0
	T_{Sched}	0
T_{Idle}	$T_{Wait_D}(p)$	$t_p^1 + \sum_{i=2}^{x_p} max\{t_p^i - t_p^{i-1} + f(\gamma_p^{i-1}), \tau\}$
	$T_{Wait_S}(p)$	$\mathcal{M}(X(c,\Gamma,\{p\},Z_0^+)) - \sum_{(\gamma,p,t)\in X(b,\Gamma,\{p\},Z_0^+)} f(\gamma) - T_{Wait_D}(p)$
	$T_{Fin}(p)$	$\mathcal{M}(c) - \mathcal{M}(X(c,\Gamma,\{p\},Z_0^+))$

Table 3–3: A *Uniform Delay* Scheduling Model in Terms of our Framework

Chapter 4

Analysis of Scheduling Algorithms

4.1 Approximation Algorithms for Scheduling Problems

As a result of the complexity of scheduling problems, it is natural to consider polynomial time algorithms which, although they are not guaranteed to find the best solution, are guaranteed to find solutions which are close to the optimum. Such algorithms are known as approximation algorithms, or in the case where there exists some positive constant ϵ such that the algorithm's solutions are guaranteed to be at most $1 + \epsilon$ times the optimal solution, as ϵ-approximation algorithms. An early partial review of this area can be found in Garey *et al.* [1977]. More recent work is described in recent reviews of scheduling: namely Cheng and Sin [1990] and Veltman *et al.* [1990].

4.1.1 No-Communication Models

Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a *No Communication* scheduling model where $|P| = n$ and $|\Gamma| = l$. Graham's Longest Processing Time (LPT) algorithm [Graham, 1969] guarantees to find a schedule s such that

$$\mathcal{M}(s) \leq (4/3 - 1/3n)\mathcal{M}(s_{opt})$$

Where s_{opt} is a valid schedule for A with the shortest possible makespan. Coffmann *et al.* [1978] give an algorithm based on techniques from bin packing and improve this to $1.22\mathcal{M}(s_{opt})$. Sahni [1976] produced a family of approximation algorithms for any guaranteed performance ϵ, whose running time was polynomial in l but exponential in n. More recently, Hochbaum and Shmoys [1988a, 1988b] have developed a family of ϵ-approximation algorithms for which the complexity is polynomial in both l and n.

As an extension to Graham's work, Coffman *et al.* [1984] consider the expected makespan for LPT scheduling under the assumption that tasks' execution times are independent, identically-distributed, random variables. They show that LPT makespans converge stochastically to optimal makespans for a range of distributions of task execution times.

4.1.2 No-Delay Models

Algorithms for solving or approximately solving the mapping problem for *No Delay* scheduling models have been appearing in the literature with remarkable frequency. In the cases where the problem is NP complete, heuristic approaches have often been used. This version of the mapping problem has been the subject of previous reviews (see Chen and Liu [1975] for a discussion of various similar heuristic approaches). Indeed it is often considered in the same reviews as *No Communication* models, and as a result we only describe a selection of the results and heuristics.

Hu [1961] showed that for UET models where the task graph is a tree, a schedule based upon sequential processing of layers of the tree is optimal. Kaufmann [1974] extended Hu's algorithm to the case of non-unit length tasks, and showed bounds on its performance which allowed him to consider it "almost optimal". Graham [1966, 1969] extended this work to propose a general technique for scheduling known as *list scheduling*. Coffman and Graham [1972] showed an algorithm which generates optimal schedules for any UET, 2 processor *No Delay* scheduling model. Their algorithm may be generalised to models with an arbitrary number of processors but it loses its exactness. Lam and Sethi [1977] showed that Coffman and Graham's algorithm will generate a schedule s where $\mathcal{M}(s)/\mathcal{M}(s_{opt}) \leq 2 - 2/n$. Gabow [1988] showed a linear time algorithm for scheduling on two *uniform* processors (recall our definition in Section 1.2.2), again with unit length tasks, which generates optimal res-

ults with certain fixed ratios of processor speeds and nearly optimal results otherwise. Cho and Sahni [1980] give bounds for list schedules on *No delay* uniform-processor models. Cole and Vishkin [1988] show logarithmic time parallel implementations of list scheduling on an EREW PRAM. There are also a number of results for preemptive scheduling of precedence constrained jobs (e.g. Muntz and Coffman. [1969] for 2-processor systems), and in this context we refer the reader to the reviews referred to in Section 3.3 and to Lawler [1982].

4.1.3 General, Uniform and Fixed Delay Models

In the later sections of this chapter we concentrate on two algorithms which were proposed by Papadimitriou and Yannakakis [1990] on the one hand and by Hwang *et al.* [1989] on the other. The latter algorithm is known as ETF. We refer to the former algorithm as PNY for the sake of brevity.

PNY works with a UET *Uniform Delay* scheduling models[1]; with what they refer to as *unbounded* numbers of processors; and performs recomputation. We have shown above the number of processors that can usefully be used by any such algorithm is bounded by $|\Gamma|$, and as we show below there is another, often tighter, processor bound for PNY. On the positive side PNY's performance is shown by Papadimitriou and Yannakakis to produce schedules which have a makespan within a factor of 2 of optimal with respect to their model. We give detailed consideration to this bound in later sections.

The unbounded processor assumption is also made by Sarkar (in the *Internalisation* phase of his work), by Yang and Gerasoulis [1992][1993] and by Kim and Browne [1988]. However none of the algorithms that these authors

[1]There is another algorithm in [Papadimitriou and Yannakakis 1990] which relaxes these two constraints

propose allow task replication. Jung *et al.* [1989] showed for the inverse binary tree that recomputation can gain performance which is a function of the interprocessor communications delay, which implies that there can be no fixed bound on the ratio by which these algorithms exceeds the optimal makespan of the computation. As we will see, determining how to remove recomputation from a Papadimitriou and Yannakakis schedule so as to minimise the number of processors being used is itself NP complete.

ETF is similar in flavour to Graham's List scheduling algorithm [Graham 1969]. It works with arbitrary length tasks; general communication delays[2]; a fixed number of processors and forbids recomputation. The result of Jung *et al.* implies that there can be no fixed bound on the ratio by which ETF exceeds the optimal makespan of the computation, however Hwang *et al.* show their algorithm satisfies a dag-dependent bound which is presented and discussed below. Scheduling heuristics for this model can also be found in Williams [1983] and Baxter and Patel [1989].

4.2 Papadimitriou and Yannakakis' Algorithm (PNY)

Given a dag $\Lambda = (\Gamma, \Delta)$ and an integer communications delay τ Papadimitriou and Yannakakis [1990] compute a function $e : \Gamma \rightarrow Z_0^+$, inductively on the depth of a node in the dag according to the following algorithm.

Algorithm 4.1 e-value

1 let $\Lambda = (\Gamma, \Delta)$ be a directed acyclic graph and τ be some integer.

2 let $\gamma_1, \gamma_2, \ldots, \gamma_m$ be an enumeration of Γ such that if $1 \leq a < b \leq m$, then $\gamma_b \notin pred(\gamma_a, \Lambda)$;

---

[2]As discussed in later sections it is not presented in this way

```
3   for i := 1 to m
    do
4       if |pred(γ_i, Λ)| = 0 then let e(γ_i) = 0;
        else do
5           let (δ_1, δ_2, ..., δ_ρ) be an enumeration of pred(γ_i, Λ) such that
            e(δ_1) ≥ e(δ_2) ≥ ... ≥ e(δ_ρ);
6           let k = min{τ + 1, ρ};
7           let e(γ_i) = e(δ_k) + k;
        enddo
    enddo
    end
```

Papadimitriou and Yannakakis prove the following properties relating to their function e and the class of schedules.

Lemma 4.1 *There is no valid schedule in which vertex γ_i is scheduled before time $e(\gamma_i)$.*

Lemma 4.2 *For each vertex γ_i there is a valid schedule in which vertex γ_i is scheduled at start time $2e(\gamma_i)$.*

The proofs of both Lemma 4.1 and Lemma 4.2 are found in [Papadimitriou and Yannakakis 1990].

4.2.1 A PNY schedule using $|\Gamma|$ processors

Using the proof of Lemma 4.2 Papadimitriou and Yannakakis suggest the following simple algorithm for scheduling the vertices of a dag, so that every vertex is executed by time $2e(\gamma_i)$. The algorithm thus first computes all e-values

then creates a schedule so that γ_i is computed at start time equal to twice its e-value.

In the algorithm below each vertex γ_i is computed on a separate processor together with the τ highest (in e value) predecessors of γ_i and receives the rest of its inputs in the form of communications from other processors. If $| pred(\gamma_i, \Lambda) | < \tau$, then all predecessors are computed on the processor.

Algorithm 4.2 Papadimitriou and Yannakakis Mark 1

1 let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a *Fixed Delay* UET Scheduling Model where $|\Gamma| = |P| = l$; and the range of d is $\{\tau, 0\}$;

2 let $\gamma_1, \gamma_2, \ldots, \gamma_l$ be an enumeration of Γ;

3 let $p_1, p_2, \ldots, p_l$ be an enumeration of P;

4 let $s = \{\}$;

5 for $a := 1$ to l

do

6 let $R_a = (\delta_1^a, \delta_2^a, \ldots, \delta_{\sigma_a}^a)$ be an enumeration of $pred(\gamma_a, \Lambda) \cup \{\gamma_a\}$ such that $\forall 1 \leq i < \sigma_a$, $e(\delta_i^a) \leq e(\delta_{i+1}^a)$ and for all $1 \leq i < j \leq \sigma_a$, $\delta_j^a \notin pred(\delta_i^a, \Lambda)$;

7 let $\rho_a = min(\tau, |pred(\gamma_a, \Lambda)|)$;

8 for $h = \sigma_a - \rho_a$ to σ_a

do

9 let $s = s \cup \{(\delta_h^a, p_a, 2e(\gamma_a) - \sigma_a + h)\}$;

enddo

enddo

enddo

end

Combining Lemmas 4.1 and 4.2 we have the following theorem from Papadimitriou and Yannakakis .

Theorem 4.1 *Algorithm 4.2 is an approximation algorithm for minimising the make-span with a worst case ratio of two.*

The use of l processors by Algorithm 4.2 is consistent with the bound of Corollary 3.2, and the correspondence can be explained by the fact that for all $1 \leq i \leq l$, each processor p_i ends by computing a distinct task γ_i. However, it is important to note that the schedule of Algorithm 4.2 can often be transformed to use fewer processors than l.

4.2.2 The Problem of PNY Pruning

There is an implicit assumption in the model underlying Papadimitriou and Yannakakis' mapping problem that processors are plentiful. Like schedules produced by Algorithm 3.1, PNY schedules use exactly as many processors as there are tasks to be scheduled. We now consider the problem of pruning a PNY schedule so that it uses a number of processors which is different from the number of tasks. We show the following problem is NP complete.

Decision Problem 4.1 PNY Pruning
Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a UET Fixed Delay scheduling model where $|\Gamma| = |P| = l$ and the range of d is $\{0, \tau\}$. Let s be a schedule for A that was generated by Algorithm 4.2. Let J be an integer. Does there exist a schedule $s' \subset s$ which is valid for A and such that $\mathcal{P}(s') \leq J$?

We reduce from a general instance of Three Minimum Cover (TMC)–Decision Problem 4.2 below, which is known to be NP complete [Garey and Johnson 1979].

Decision Problem 4.2 TMC
Let $S = \{S_1, \ldots, S_a\}$ be a set of items. Let $C = \{C_1, \ldots C_b\}$ be a collection of subsets

of S such that $\mid C_i \mid = 3$, $\forall i = 1, \ldots, b$ and $\bigcup_{i=1}^{b} C_i = S$. Let $K < b$ be a positive integer. Does there exist a cover *of S, C', such that $\mid C' \mid \leq K$? That is, does there exist a subset $C' \subset C$ such that, $\bigcup_{C_i \in C'} C_i = S$?*

We shall need the following lemma.

Lemma 4.3 *Let S, C, K be respectively a set of items, a collection of subsets and a positive integer forming an instance of TMC. Let C' be a collection of subsets of S. Let $K' = \mid C' \mid$ and let S' be a subset of S such that $(\bigcup_{C_i \in C'} C_i) \cup S' = S$. Let $J' = \mid S' \mid$. S has a cover of size no more than $K' + J'$.*

Proof
Let $S_1, S_2, \ldots, S_a$ be an enumeration of S'. For each $S_i \in S'$ choose some $\hat{C}_i \in C$ such that $S_i \in \hat{C}_i$. Let $C^* = (\bigcup_{i=1}^{a} \{\hat{C}_i\}) \cup C'$. C^* is the cover as required. □

We now define a polynomial time encoding from TMC to PNY pruning.

Definition 4.1 TMC to PNY Pruning encoding.
Let S, C, K respectively be a set of items, a collection of subsets and a positive integer forming an instance of TMC. Let $S_1, S_2, \ldots, S_a$ be an enumeration of S and let $C_1, C_2, \ldots, C_b$ be an enumeration of C. We construct our instance of Decision Problem 4.1 in the following way. The resulting dag is shown in Figure 4–1.

$$
\begin{aligned}
\Upsilon &= \{v_j^i : j = 1, \ldots, b,\ i = 1, \ldots, 3\} \\
\Phi &= \{\phi_j^i : j = 1, \ldots, b,\ i = 1, \ldots, 3\} \\
\Xi &= \{\xi_j^i : j = 1, \ldots, b,\ i = 1, \ldots, 3\} \\
\Psi &= \{\psi_j : j = 1, \ldots, b\} \\
\Gamma &= S \cup C \cup \Upsilon \cup \Phi \cup \Xi \cup \Psi \\
\Delta &= \{(\xi_j^i, v_j^i) : i = 1 \ldots 3, j = 1 \ldots b\} \quad \cup
\end{aligned}
$$

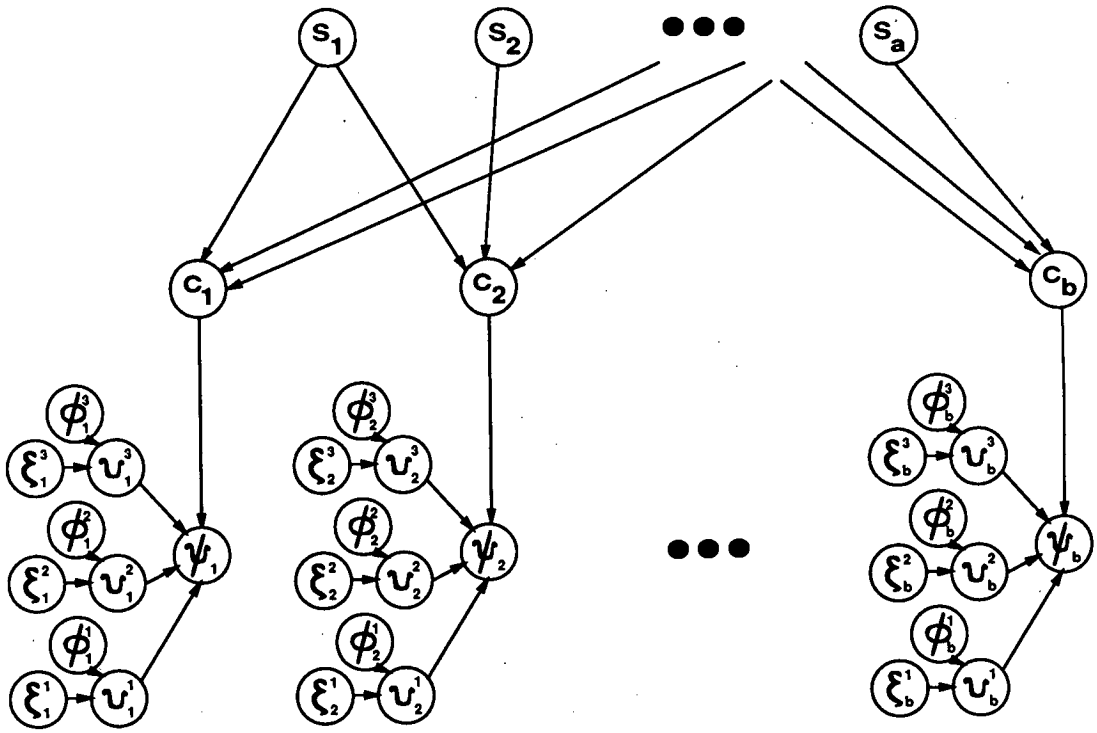


Figure 4–1: The Constructed dag Obtained in the Polynomial Time Reduction from Three Minimum Cover.

$$
\begin{aligned}
& \{(\phi_j^i, \upsilon_j^i) : i = 1 \ldots 3, j = 1 \ldots b\} \quad \cup \\
& \{(\upsilon_j^i, \psi_j) : i = 1 \ldots 3, j = 1 \ldots b\} \quad \cup \\
& \{(C_i, \psi_i) : i = 1 \ldots b\} \quad \cup \\
& \{(S_i, C_j) : S_i \in C_j\} \\
\tau = \quad & 3 \\
P = \quad & \{p_1, p_2, \cdots p_{|\Gamma|}\} \\
J = \quad & 4b + K
\end{aligned}
$$

We note the following properties of our encoded instance. Papadimitriou and Yannakakis assign the following e-values to the vertices in Γ: $e(S_i) = 0$ for all $S_i \in S$; $e(\phi) = 0$ for all $\phi \in \Phi$; $e(\xi) = 0$ for all $\xi \in \Xi$; $e(\upsilon) = 2$ for all $\upsilon \in \Upsilon$; $e(C_i) = 3$ for all $C_i \in C$; $e(\psi) = 6$ for all $\psi \in \Psi$.

Let $\gamma_1, \gamma_2, \ldots, \gamma_l$ be the enumeration of Γ and $(p_1, p_2, \ldots, p_l)$ be the enumeration of P chosen by Algorithm 4.2, Application of Algorithm 4.2 results in the schedule shown in the Gantt chart of Figure 4–2. Processor p_i is assigned task γ_i and its $\rho_i = \min\{\tau, |pred(\gamma_i, \Lambda)|\}$ highest in e-value predecessors. In the following we shall make much use of the correspondence between the indices in the enumerations of P and Γ. Below, we identify the predecessors that are executed with each task. Where there is a non-deterministic choice to be made by Algorithm 4.2 we express that non-determinism.

For all $\gamma_i \in S \cup \Phi \cup \Xi$, $pred(\gamma_i, \Lambda) = \{\}$ and so

$$\forall \gamma_i \in S \cup \Phi \cup \Xi, \quad X(s, \Gamma, \{p_i\}, Z_0^+) = \{(\gamma_i, p_i, 0)\}.$$

For all $\gamma_i = C_j \in C$, $pred(C_j, \Lambda) = \{s \in S : s \in C_j\}$. Thus $|pred(C_j, \Lambda)| = 3 = \tau$. Let (S_j^1, S_j^2, S_j^3) be the order in which $pred(C_j, \Lambda)$ were arranged by Algorithm 4.2, then

$$\forall \gamma_i = C_j \in C, \quad X(s, \Gamma, \{p_i\}, Z_0^+) = \bigcup_{k=1,2,3} \{(S_j^k, p_i, 6-k)\} \cup \{(C_j, p_i, 6)\}.$$

For all $\gamma_i = v_j^k \in \Upsilon$, $pred(v_j^k, \Lambda) = \{\phi_j^k, \xi_j^k\}$. Thus $|pred(v_j^k, \Lambda)| = 2 < \tau$. Let $t_{\phi_j^k}$ be chosen nondeterministically from $\{2, 3\}$. Let $t_{\xi_j^k} = 5 - t_{\phi_j^k}$

$$\forall \gamma_i = v_j^k \in \Upsilon, \quad X(s, \Gamma, \{p_i\}, Z_0^+) = \{(v_j^k, p_i, 4), (\xi_j^k, p_i, t_{\xi_j^k}), (\phi_j^k, p_i, t_{\phi_j^k})\}.$$

For all $\gamma_i = \psi_j \in \Psi$, $pred(\psi_j, \Lambda) = \bigcup_{k=1,2,3}\{v_j^k\} \cup \bigcup_{k=1,2,3}\{\phi_j^k\} \cup \bigcup_{k=1,2,3}\{\xi_j^k\} \cup C_i$. Note that for each C_i, $e(C_i) = 3$, for each v_j^j, $e(v_j^k) = 2$, and for all ϕ_j^k, ξ_j^k, $e(\gamma) = 0$ thus

$$\forall \gamma_i = \psi_j \in \Psi, \quad X(s, \Gamma, \{p_i\}, Z_0^+) \supset \{(C_j, p_i, 11), (\psi_j, p_i, 12)\}.$$

Finally, before our NP completeness proof, we will require to prove the following lemma which states that at least $4b$ processors are required to compute tasks which are not in $C \cup S$. Moreover, these processors all correspond (in our pairing of indices) to tasks which are not in $C \cup S$.

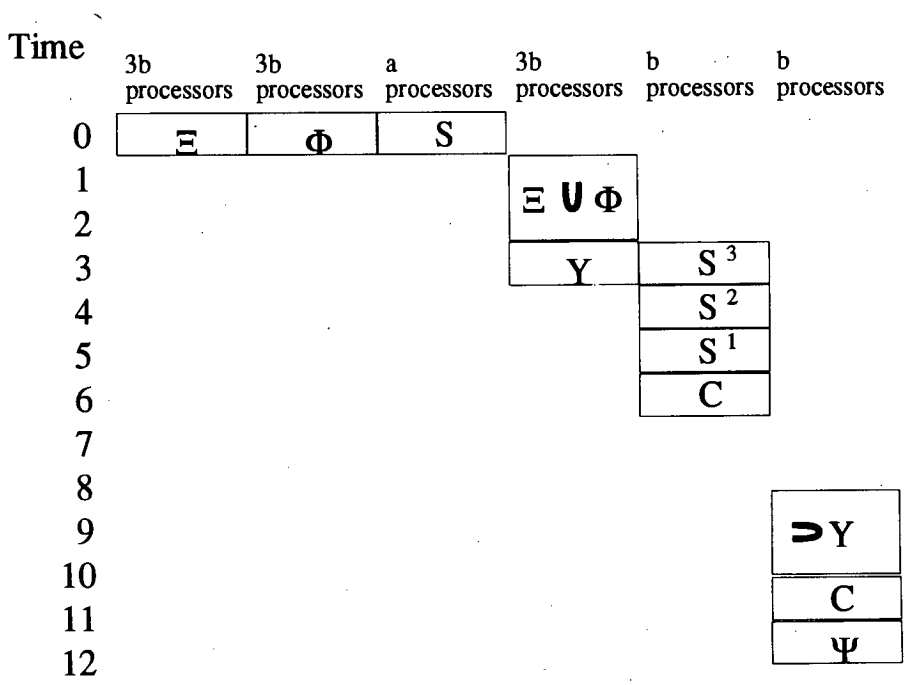


Figure 4–2: The Schedule Obtained from Algorithm 4.2 on a dag Encoded as in Definition 4.1

Lemma 4.4 *Let* $A = (P, \Lambda = (\Gamma, \Delta), f, d)$, *s and J be an instance of PNY pruning which has been constructed from an instance* S, C, K *of TMC as described in Definition 4.1. Let* $\Gamma^- = \Gamma - S$. *Let* $\Delta^- = \{(\gamma, \delta) \in \Delta : \gamma, \delta \in \Gamma^-\}$. *If a schedule s is valid for* $B = (P, (\Gamma^-, \Delta^-), f, d)$ *then* $\mathcal{P}(X(s, \Gamma, \{p_i \in P : \gamma_i \in \Gamma - (S \cup C)\}, Z_0^+)) \geq 4b$.

Proof

Let us assume as a hypothesis to be proven contradictory that there exists a schedule s which is valid for B and such that $\mathcal{P}(X(s^*, \Gamma, \{p_i \in P : \gamma_i \in \Gamma - (S \cup C)\}, Z_0^+)) < 4b$. Since $|\Phi| = |\Upsilon| = 3b$ and $|\Psi| = b$ this implies one or other of the following.

- There exists some task $\gamma_i = \psi_j \in \Psi$ such that p_i is idle in s. In this case ψ_j is never executed which leads to a contradiction.

- There exists some pair of tasks $\gamma_i = \upsilon_i^j \in \Upsilon$ and $\gamma_h = \phi_i^j \in \Phi$ such that both p_i and p_h are idle in s. In this case ϕ_i^j is never executed which leads to a contradiction.

□

Theorem 4.2 *The problem of PNY pruning is NP complete.*

Proof

Let s be a schedule produced by Algorithm 4.2 on an instance $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ of a *Unit Delay* scheduling model. We can check if a given schedule $s' \subset s$ is valid by checking that it is *complete* and neither *temporally compromised* nor *overbooked*

To check that it is *complete* we need to check that for all $\gamma \in \Gamma$,

$$X(s', \{\gamma\}, P, Z_0^+) \neq \{\},$$

which can be achieved by searching $|\Gamma|$ times through $|s'|$ tuples.

To check that it is not *overbooked* we need to check that there exists no tuple $(\gamma, p, t) \in s'$ such that $X(s', \Gamma - \{\gamma\}, \{p\}, \{t, t+1, \ldots, t+f(\gamma)-1\}) \neq \{\}$. This can be achieved by searching $|s'|$ times through $|s'|$ tuples.

To check that it is not *temporally compromised* we need to show that there exists no tuple $\rho = (\gamma, \delta) \in \Delta$ such that there exists a tuple $(\delta, p, t) \in s'$ such that $\min_{(\gamma,q,t') \in s'}(t' + f(\gamma) + d(\rho, p, q)) > t$. This can be achieved by searching $|s'|$ times through $|s'|$ tuples for each of $|\Delta| \leq |\Gamma|^2$ edges.

Since $s' \subset s$, we can conclude that the time complexity of checking the validity of the schedule is at most $O(|s|^2|\Gamma|^2)$, which is a polynomial in the size of our instance. Since we can just guess non-deterministically a pruning of s such that $\mathcal{P}(s) \leq J$ and test that it is valid, we have established that the PNY pruning problem is in NP.

Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$, s and J be an instance of PNY pruning which has been constructed from an instance S, C, K of TMC as described in Definition 4.1. We now show that s can be pruned to a valid schedule s' with fewer than J non-idle processors if and only if there is a subset $C' \subseteq C$, $|C'| \leq K$, which covers S.

if Let $C' \subset C$ be a cover of size K of the set S. Let $C'_1, C'_2, \ldots, C'_K$ be an enumeration of C'. We prune our schedule as follows:

$$
\begin{aligned}
s' &= X(s, \Gamma, \{p_i \in P : \gamma_i \in \Upsilon\}, Z_0^+) \cup \\
&\quad X(s, C \cup \Psi, \{p_i \in P : \gamma_i \in \Psi\}, Z_0^+) \cup \\
&\quad X(s, S, \{p_i \in P : \gamma_i \in C'\}, Z_0^+) \\
&= \bigcup_{\gamma_i = v_j^k,\ j=1,\ldots,b,\ k=1\ldots,3} \{(v_j^k, p_i, 4), (\xi_j^k, p_i, t_{\xi_j^k}), (\phi_j^k, p_i, t_{\phi_j^k})\} \cup \\
&\quad \bigcup_{\gamma_i = \psi_j,\ j=1,\ldots,b,} \{(C_j, p_i, 11), (\psi_j, p_i, 12)\} \cup \bigcup_{\gamma_i = C_j \in C} \bigcup_{k=1,2,3} \{(S_j^k, p_i, 6-k)\}.
\end{aligned}
$$

Note $\mathcal{P}(s') = |\Upsilon| + |\Psi| + |C'| = 3b + b + K = J$. Note that for all tasks $\gamma \in \Upsilon \cup \Phi \cup \xi \cup \Psi \cup C$, we have that $X(s', \{\gamma\}, P, Z_0^+) \neq \{\}$. Moreover since C' covers S,

$$\bigcup_{\gamma_i = C_j \in C} \bigcup_{k=1,2,3} \{(S_j^k, p_i, 6-k)\}$$

contains a tuple instancing every $S_i \in S$. Thus we conclude s' is *complete.*

Note also that $s' \subseteq s$ and s is valid, and thus by Lemma 3.2 s' is not overbooked

Let us assume as a hypothesis to be proven contradictory that s' is temporally compromised. This implies there exists a tuple (γ_i, p_i, t) such that one of the following is true.

- $\gamma_i = v_j^k \in \Upsilon$ and ϕ_j^k is not executed on processor p_i before time $t = 3$, which leads to a contradiction.
- $\gamma_i = v_j^k \in \Upsilon$ and ξ_j^k is not executed on processor p_i before time $t = 3$, which leads to a contradiction.
- $\gamma_i = \psi_j \in \Psi$ and there exists some $v_j^k \in \Upsilon$ which is not executed on any processor before time $t' = t - \tau = 12 - 3 = 9$, which leads to a contradiction.
- $\gamma_i = \psi_j \in \Psi$ and C_j is not executed on processor p_i before time $t = 12$, which leads to a contradiction.
- $\gamma_i = C_j \in C$ and there exists a task $S_k \in S$ such that S_k is not executed before time $t' = t - \tau = 11 - 3 = 8$, which leads to a contradiction.

Thus we conclude that s' is complete and neither temporally compromised nor overbooked and thus valid for A and we further conclude there is a valid pruning of s with $\mathcal{P}(s') \leq J$ if there exists a K cover for S.

only if Let $s^* \subseteq s$ be a valid schedule with $\mathcal{P}(s^*) \leq J = K + 4b$. We shall establish there must exist a K cover for S. Let $s' = X(s^*, S, P, Z_0^+)$. Let $C' = \{\gamma_i \in C : X(s', S, \{p_i\}, Z_0^+) \neq \{\}\}$. Let $S' = \{\gamma_i \in S : X(s', S, \{p_i\}, Z_0^+) \neq \{\}\}$.

Note $X(s, S, \{p_i \in P : \gamma_i \in \Gamma - (S \cup C)\}, Z_0^+) = \{\}$ and by Lemma 4.4 $\mathcal{P}(X(s^*, \Gamma, \{p_i \in P : \gamma_i \in \Gamma - (S \cup C)\}, Z_0^+)) \geq 4b$. Thus $\mathcal{P}(s') \leq K$ and $|C'| + |S'| \leq K$. Let us assume that there exists some task $S_j \in S$ such that $S_j \notin \cup_{C_i \in C'} C_i \cup S'$. Now s^* is complete thus S_j must be executed. We have two cases

- $S_j \in S'$ which leads immediately to a contradiction.
- There exists a task $\gamma_i = C_h \in C'$ such that $X(s', \{S_j\}, \{p_i\}, Z_0^+) \neq \{\}$ which implies $S_j \in pred(C_h, \Lambda)$ and so by construction $S_j \in C_h$, which leads to a contradiction.

Thus $\cup_{C_i \in C'} C_i \cup S' = S$ and so, by Lemma 4.3, S has a cover of at most size $|S'| + |C'| = K$ and the theorem is proved. □

Note that the above argument holds irrespective of the non-determinism in Algorithm 4.2. That is, we are not concerned with the ordering of the recomputation of the predecessors of any task, nor by the subset of the predecessors of each ψ which get computed with it.

4.2.3 A Slightly Less Processor Wasteful Algorithm

We now know it is an NP complete problem to determine whether it is possible to prune a PNY schedule so as to use at most a given number of processors. Consider the algorithm below, which is a modification of Algorithm 4.2, which leaves out the computation on certain processors. Specifically, we prune all computation mapped to a processor p_i where for the corresponding task γ_i, there exists some successor γ_j such that $e(\gamma_i) = e(\gamma_j)$. The rather complex proof of Theorem 4.3 establishes that this computation is redundant.

Schedules generated by this algorithm are used in later sections where we show bounds on the number of active processors and indicate how they can be transformed to schedules with fewer processors, in some cases with the same performance guarantee, or in general with a weaker performance guarantee.

Algorithm 4.3 Papadimitriou and Yannakakis Mark 2

1 let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a *Fixed Delay* UET Scheduling Model where $|\Gamma| = |P| = l$; and the range of d is $\{\tau, 0\}$;

2 let $\gamma_1, \gamma_2, \ldots, \gamma_l$ be an enumeration of Γ;

3 let $p_1, p_2, \ldots, p_l$ be an enumeration of P;

4 let $s = \{\}$;

5 for $a := 1$ to l

do

6 if $\{\alpha \in succ(\gamma_a) : e(\alpha) = e(\gamma_a)\} = \{\}$

do

7 let $R_a = (\delta_1^a, \delta_2^a, \ldots, \delta_{\sigma_a}^a)$ be an enumeration of $pred(\gamma_a, \Lambda) \cup \{\gamma_a\}$ such that $\forall 1 \leq i < \sigma_a$, $e(\delta_i^a) \leq e(\delta_{i+1}^a)$ and for all $1 \leq i < j \leq \sigma_a$, $\delta_j^a \notin pred(\delta_i^a, \Lambda)$;

8 let $\rho_a = min(\tau, |pred(\gamma_a, \Lambda)|)$;

9 for $h = \sigma_a - \rho_a$ to σ_a

do

10 let $s = s \cup \{(\delta_h^a, p_a, 2e(\gamma_a) - \sigma_a + h)\}$;

enddo

enddo

enddo

end

Before we can show the validity of schedules produced by the algorithm, we first need to show that in its schedules, every task is executed at least once by a time which is twice its e-value.

Lemma 4.5 *Let* $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ *be an instance of a UET Fixed Delay scheduling model. Let* s *be a schedule generated by Algorithm 4.3 for* A. *For all* $\gamma \in \Gamma$, $X(s, \{\gamma\}, P, \{0, 1, \ldots, 2e(\gamma)\}) \neq \{\}$.

Proof

Let us assume as a hypothesis to be proven contradictory that there exists some task γ_i such that $X(s, \{\gamma_i\}, P, \{0, 1, \ldots, 2e(\gamma_i)\}) = \{\}$. If the condition at Line 6 of Algorithm 4.3 holds for γ_i then $X(s, \{\gamma_i\}, P, \{0, 1, \ldots, 2e(\gamma_i)\}) \supseteq \{(\gamma_i, p_i, 2e(\gamma_i))\}$, so where

$$S = \{\alpha \in succ(\gamma_i) : e(\alpha) = e(\gamma_i)\},$$

we shall assume that $S \neq \{\}$.

Let $\gamma_j \in S$ be a task such that

$$\{\beta \in succ(\gamma_j) : e(\beta) = e(\gamma_j)\} = \{\}.$$

We note that γ_j must exist because S is finite and Λ is acyclic. Note that the condition at Line 6 of Algorithm 4.3 holds for γ_j, and thus by Line 10, for each $\gamma \in (\delta_{\sigma_j - \rho_j}, \ldots \delta_{\sigma_j}), X(s, \{\gamma\}, P, \{0, 1, \ldots, 2e(\gamma)\}) \neq \{\}$ Thus we may assume that, notwithstanding $\gamma_i \in pred(\gamma_j, \Lambda)$, $\gamma_i \notin (\delta_{\sigma_j - \rho_j}, \ldots, \delta_{\sigma_j})$.

We have two cases,

1. $|pred(\gamma_j, \Lambda)| \leq \tau$ in which case since $\gamma_i \in pred(\gamma_j, \Lambda)$, we know that $|pred(\gamma_i, \Lambda)| < |pred(\gamma_j, \Lambda)|$ and thus $e(\gamma_i) = |pred(\gamma_i, \Lambda)| < |pred(\gamma_j, \Lambda)| = e(\gamma_j)$ which is contradictory to our construction of S.

2. $|pred(\gamma_j, \Lambda)| > \tau$ thus by Algorithm 4.1, $e(\gamma_j) > e(\gamma_i) + \tau$, which is again contradictory to our construction of S.

Thus we conclude that in all cases that our hypothesis is contradictory, and the lemma is proved. □

Since every task is executed by a time equal to twice its e-value, every task is executed at least once, and so we have the following simple corollary to Lemma 4.5.

Corollary 4.1 *Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a UET Fixed Delay scheduling model. Let s be a schedule generated by Algorithm 4.3 for A. s is complete.*

Next we show that Algorithm 4.3 generates valid schedules.

Theorem 4.3 *Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a UET Fixed Delay scheduling model. Let s be a schedule generated by Algorithm 4.3 for A. s is valid.*

Proof
By Corollary 4.1, s is complete. Line 10 adds tuples of the form $(\gamma, p_a, 2e(\gamma_a) - \sigma_a + h)$ where $\gamma, \gamma_a \in \Gamma$. Note that a and h uniquely identify an execution of Line 10 of Algorithm 4.3, and thus it is not possible for two tuples to be inserted into s with both an identical processor and an identical execution time. Thus we conclude s is not overbooked.

Let $\gamma_1, \gamma_2, \ldots, \gamma_l$ be the enumeration of Γ and $p_1, p_2, \ldots, p_l$ be the enumeration of P chosen by Algorithm 4.3. Let us assume as a hypothesis to be proven contradictory that s is temporally compromised. Note that A is an instance of a UET scheduling model. This would imply there existed some edge, say $(\gamma_k, \gamma_j) \in \Delta$ such that there existed some tuple, say $(\gamma_j, p_i, t) \in s$ where p_j and p_i (the processor corresponding to γ_i in the pair of enumerations) are not necessarily distinct, such that, defining

$$\hat{t} = \min_{(\gamma_k, p', t') \in X(s, \{\gamma_k\}, P, Z_0^+)} t' + d((\gamma_k, \gamma_j), p', p_i),$$

we have that $\hat{t} \geq t$.

Note $\gamma_k \in pred(\gamma_j, \Lambda)$ and since γ_j is executed on p_i, by Line 7 of Algorithm 4.3, $\gamma_j \in pred(\gamma_i, \Lambda) \cup \{\gamma_i\}$, thus $\gamma_k \in pred(\gamma_i, \Lambda)$. Note also that by Line 10 of Algorithm 4.3, $t \leq 2e(\gamma_i) - \tau$.

We have two cases:

1. $X(s, \{\gamma_k\}, \{p_i\}, Z_0^+) \neq \{\}$. Thus there must be a tuple, say $(\gamma_k, p_i, t^*) \in X(s, \{\gamma_k\}, \{p_i\}, Z_0^+)$. Since $\gamma_k \in pred(\gamma_i, \Lambda)$, by Line 10 of Algorithm 4.3, $t^* < t$. But by Definition 2.22 $d((\gamma_k, \gamma_j), p_i, p_i) = 0$. Thus $\hat{t} \leq t^* < t$, which leads to a contradiction.

2. $X(s, \{\gamma_k\}, \{p_i\}, Z_0^+) = \{\}$, in which case since $\gamma_k \in pred(\gamma_i, \Lambda)$, $e(\gamma_k) \leq e(\gamma_i) - \tau - 1$, and by Lemma 4.5,

$$X(s, \{\gamma_k\}, P, \{0, 1, \ldots, 2(e(\gamma_i) - \tau - 1)\}) \neq \{\}.$$

   Furthermore the range of d is upper bounded by τ, thus $\hat{t} \leq 2(e(\gamma_i) - \tau - 1) + \tau$, which means $\hat{t} \leq t - 2$ which leads to a contradiction.

Thus we conclude that in all cases s is not temporally compromised. And since we have concluded that s is complete and neither temporally compromised nor overbooked we conclude it is valid and the theorem is proved. □

4.2.4 A PNY Schedule using $\mathcal{W}(\Lambda)\lceil\frac{\tau+1}{2}\rceil$ Processors

We have established that Algorithm 4.3 generates valid schedules. We now consider the way in which these schedules can be made less processor-greedy by *remapping*. First we consider how many processors are in use at any one time in the schedules produced by Algorithm 4.3. Next we consider putting these schedules through Algorithm 3.2, making use of various lemmas proved in Section 3.2.

We will need the following lemmas.

Lemma 4.6 *For Papadimitriou and Yannakakis' function e, computed on a task γ in a dag Λ such that $|pred(\gamma, \Lambda)| \leq \tau$, $e(\gamma) = |pred(\gamma, \Lambda)|$.*

Proof

We consider the state of the variables in Algorithm 4.1 at the assignment of some task γ such that $|pred(\gamma, \Lambda)| \leq \tau$. At Line 5 $\rho = |pred(\gamma)|$, and by the enumeration at Line 5, $|pred(\delta_\rho)| = 0$. By the enumeration at Line 2 and the assignment at Line 4, the value $e(\delta_\rho) = 0$ has already been assigned. By Line 6, $k = \rho$ and thus by Line 7 $e(\gamma_i) = e(\delta_\rho) + \rho = 0 + |pred(\gamma, \Lambda)|$. □

Lemma 4.7 *Papadimitriou and Yannakakis' function e, computed on a dag Λ is non-decreasing along any path in Λ.*

Proof

Let $\Lambda = (\Gamma, \Delta)$ be a dag.

Let us assume as a hypothesis to be proven contradictory that there exists a path, say $\gamma_1, \gamma_2, \ldots, \gamma_m$ in Λ such that the function e is decreasing along the path. Thus there exists an edge, say $(\gamma_{i-1}, \gamma_i) \in \Delta$ such that $e(\gamma_{i-1}) > e(\gamma_i)$. We have two cases

- $e(\gamma_i) = |pred(\gamma_i, \Lambda)| \leq \tau$ in which case by Lemma 4.6,

  $$e(\gamma_{i-1}) = |pred(\gamma_{i-1}, \Lambda)| < |pred(\gamma_i, \Lambda)| = e(\gamma_i)$$

  which leads to a contradiction.

- $|pred(\gamma_i, \Lambda)| > \tau$ in which case let $\delta_1, \delta_2, \ldots, \delta_n$ be an enumeration of $pred(\gamma_i)$ such that for $j = 2, \ldots, n$ $e(\delta_j) \geq e(\delta_{j-1})$. Now $e(\gamma_i) = e(\delta_{\tau+1}) + \tau + 1$ and since $(\gamma_{i-1}, \gamma_i) \in \Delta$. $pred(\gamma_{i-1}, \Lambda) \subset pred(\gamma_i, \Lambda)$ We have two subcases

  - $|pred(\gamma_{i-1}, \Lambda)| \leq \tau$. Thus by Lemma 4.6, $e(\gamma_{i-1}) \leq \tau + 1 \leq e(\gamma_i)$ which leads to a contradiction.

- $|pred(\gamma_{i-1}, \Lambda)| > \tau$. Thus $e(\gamma_{i-1}) \leq \tau + 1 + e(\delta_{\tau+2}) \leq \tau + 1 + e(\delta_{\tau+1}) = e(\gamma_i)$ which leads to a contradiction.

□

This leads us on to considering the set of tasks γ_i such that processor p_i is allocated computation by Algorithm 4.3. In the following lemma we state that the number of such tasks sharing the same e value is at most the width of the dag.

Lemma 4.8 *Let $\Lambda = (\Gamma, \Delta)$ be a directed graph. For each $\gamma \in \Gamma$ we use Papadimitriou and Yannakakis' definition of $e(\gamma)$. Let $\Gamma' = \{\gamma \in \Gamma : \forall \delta \in succ(\gamma)\ e(\delta) \neq e(\gamma)\}$. Let Φ be a subset of Γ' such that for all $\gamma, \delta \in \Phi$, $e(\gamma) = e(\delta)$. $|\Phi| \leq \mathcal{W}(\Lambda)$.*

Proof

Let us assume that $|\Phi| > \mathcal{W}(\Lambda)$. Thus Φ is not an antichain and thus there exists some pair of vertices, say $\gamma, \delta \in \Phi$ such that there is a directed path from γ to δ. Let this path be denoted $\gamma = \delta_1, \delta_2, \ldots, \delta_m = \delta$ for some $m \geq 2$. We know $e(\gamma) = e(\delta)$ and so by Lemma 4.7 $e(\gamma) = e(\delta_1) = \ldots = e(\delta_m) = e(\delta)$. But if $e(\gamma) = e(\delta)$ and there exists a path from γ to δ then $\gamma \notin \Gamma'$ which leads to a contradiction. Hence $|\Phi| \leq \mathcal{W}(\Lambda)$ as required. □

We are now in a position to state how many processors are active at any one time in s.

Theorem 4.4 *Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a UET Fixed Delay scheduling model. Let s be a schedule generated by Algorithm 4.3.*

$$\max_{t=0}^{\mathcal{M}(s)} |\mathcal{A}(s,t)| \leq \mathcal{W}(\Lambda)\lceil(\tau+1)/2\rceil.$$

Proof

Let us assume as a hypothesis to be proven contradictory that there exists some time t such that $|\mathcal{A}(s,t)| > \mathcal{W}(\Lambda)\lceil(\tau+1)/2\rceil$. Let $p_1, \ldots, p_l$ and $\gamma_1, \ldots, \gamma_l$ be the

enumerations of P and Γ chosen by Algorithm 4.3. For $a = 0, \ldots, \mathcal{M}(s) - 1$ let us define P_a as the set of processors which, in s, finish being active at time a. That is,

$$P_a = \{p_i : 2e(\gamma_i) = a \text{ and } \forall \alpha \in succ(\gamma_i), e(\alpha) \neq e(\gamma_i)\}.$$

Note that for $a = 1, 3, \ldots, \mathcal{M}(s) - 2$, $|P_a| = 0$ and , by Lemma 4.8, for $a = 0, 2, \ldots, \mathcal{M}(s) - 1$, $|P_a| \leq \mathcal{W}(\Lambda)$. Note also that since s is not overbooked and since processors in P_a are only active from at earliest time $a - \tau$ to at latest time a, we have that

$$|\mathcal{A}(s,t)| \leq \sum_{a=max(0,t-\tau)}^{t} |P_a|.$$

Thus

$$|\mathcal{A}(s,t)| \leq \mathcal{W}(\Lambda)\lceil(\tau+1)/2\rceil$$

which leads to a contradiction. Thus we conclude that

$$\max_{t=0}^{\mathcal{M}(s)-1} |\mathcal{A}(s,t)| \leq \mathcal{W}(\Lambda)\lceil(\tau+1)/2\rceil$$

and the theorem is proved. □

Finally we consider the use of the General Purpose Remapper Algorithm defined earlier (Algorithm 3.2).

Theorem 4.5 *Let $A = (P, \Lambda, f, d)$ be an instance of a Fixed Delay UET scheduling model. Let s be the result of applying Algorithm 4.3 to A. Let s' be the result of applying Algorithm 3.2 to s. s' is a valid schedule with $\mathcal{P}(s') \leq \mathcal{W}(\Lambda)\lceil(\tau+1)/2\rceil$.*

Proof

Note that in Algorithm 4.3 task γ_i is allocated to processor p_i with ρ_i of its ancestors executed in previous time steps *in a continuous sequence.* This means that, by Lemma 3.9, s' is not temporally compromised. Moreover, by Lemma 3.8 s' is complete and not overbooked, and thus s' is valid. Furthermore, by Lemma 3.8

and Theorem 4.4,

$$\begin{aligned}\mathcal{P}(s') &\leq \max_{t=0}^{\mathcal{M}(s)-1} |\mathcal{A}(s,t)| \\ &\leq \mathcal{W}(\Lambda)\lceil(\tau+1)/2\rceil.\end{aligned}$$

□

4.2.5 A PNY Schedule Using an Arbitrary Number of Processors

Finally in this chapter, we consider mapping a PNY schedule to some arbitrary number of processors, a. We use a simple round-robin scheduler [Brent 1974] as defined below.

Definition 4.2 Round Robin Reschedule.
Let $A = (P = \{p_0, \ldots, p_{n-1}\}, \Lambda, f, d)$ be an instance of a *Fixed Delay* UET scheduling model and $a \leq n$ be a positive integer . Let s be a valid schedule for A. For $i = 0, \ldots, n-1$, let $p'_i = p_{i \bmod a}$. Let $b = \lceil n/a \rceil$. We define s' the *round robin reschedule* of s to a processors as follows.

$$s' = \{(\gamma, p'_i, bt + \lfloor i/a \rfloor) : (\gamma, p_i, t) \in s\}$$

Theorem 4.6 *Let $A = (P = \{p_0, \ldots, p_{n-1}\}, \Lambda, f, d)$ be an instance of a Fixed Delay UET scheduling model and $a \leq n$ be a positive integer. Let s be a valid schedule for A. Let s' be the round robin reschedule of s to a processors. s' is a valid schedule for A with at most a busy processors and $\mathcal{M}(s') \leq \lceil n/a \rceil \mathcal{M}(s)$.*

Proof
Let us assume as a hypothesis to be proven contradictory that for some processor p_i, $i > a$ $X(s', \Gamma, \{p_i\}, Z_0^+) \neq \{\}$. Thus, by Definition 4.2, there exists a processor q_j such that $j \bmod a > a$, which leads to a contradiction. Thus we conclude that for any processor p_i which is active in s', $1 \leq i \leq a$ and thus $\mathcal{P}(s') \leq a$.

Let (γ, p_i', t') be a tuple such that $t' + 1 = \mathcal{M}(s)$. Now there exists a tuple $(\gamma, p_i, t) \in s$ such that $t' = bt + \lfloor i/a \rfloor$, and so $\mathcal{M}(s) - 1 \geq t$. Thus $b(\mathcal{M}(s) - 1) \geq t' - \lfloor i/a \rfloor$, and since $\lfloor i/a \rfloor \leq b-1$, $b(\mathcal{M}(s)-1) \geq t' - b + 1$. Thus $b\mathcal{M}(s) \geq t' + 1 = \mathcal{M}(s')$.

Note that for every tuple $(\gamma, p_l, t) \in s$ there is a tuple $(\gamma, p_l', t') \in s'$ and thus, since s is complete, s' is complete.

Let us assume as a hypothesis to be proven contradictory that s' is temporally compromised. Since s is valid this means there exists some tuple, say $(\gamma, p_i, x) \in s$ for which there exists an edge, say $\rho = (\delta, \gamma) \in \Delta$ and a tuple $(\delta, p_j, t) \in s$ such that

$$t + 1 + d(\rho, p_j, p_i) \leq x$$

whereas, for the corresponding pair of tuples, $(\gamma, p_i', bt + \lfloor i/a \rfloor), (\delta, p_j', bt + \lfloor j/a \rfloor) \in s'$,

$$bt + \lfloor i/a \rfloor + 1 + d(\rho, p_j', p_i') > bx + \lfloor j/a \rfloor.$$

Thus

$$bt + \lfloor i/a \rfloor + 1 + d(\rho, p_j', p_i') > bt + b + bd(\rho, p_j, p_i) + \lfloor j/a \rfloor$$

and, since $\lfloor j/a \rfloor \geq 0$,

$$\lfloor i/a \rfloor + d(\rho, p_j', p_i') \geq b + bd(\rho, p_j, p_i).$$

Note that if $p_j = p_i$, $p_j' = p_i'$ and so $d(\rho, p_j, p_i) \geq d(\rho, p_j', p_i')$ and since $b \geq 1$,

$$\lfloor i/a \rfloor \geq b = \lceil n/a \rceil$$

and thus $i \geq n$ which leads to a contradiction.

Let us assume as a hypothesis to be proven contradictory that s' is overbooked. Since s is valid this means there exists some pair of tuples, say $(\gamma, p_i, t), (\gamma, p_j, t) \in s$ such that $i \neq j$ and $i \bmod a = j \bmod a$ and $bt + \lfloor i/a \rfloor = bt + \lfloor j/a \rfloor$. Thus $|i - j| < a$, which leads to a contradiction. □

We can now take the performance guarantee for Algorithm 4.2, use Algorithm 4.3, remap to fewer processors using Algorithm 3.2, and perform a round-robin schedule of the result. We finally end up with a schedule to some fixed number of processors for which we have the performance guarantee stated in the corollary below.

Corollary 4.2 *Let $A = (P = \{p_0, \ldots, p_{n-1}\}, \Lambda, f, d)$ be an instance of a Fixed Delay UET scheduling model and $a \leq n$ be a positive integer. Let s be a schedule produced by Algorithm 4.3 for A. Let s' be the result of applying Algorithm 3.2 to s. Let s^* be the Round Robin Reschedule of s' to a processors. Let s_{opt} be a valid schedule of A of minimal makespan.*

$$\mathcal{M}(s^*) \leq 2\lceil \frac{\mathcal{W}(\Lambda)\lceil(\tau+1)/2\rceil}{a} \rceil \mathcal{M} <(s_{opt})$$

4.3 ETF

4.3.1 The ETF Algorithm

Unlike PNY, ETF does not allow recomputation, so the execution of each task is performed only once and no pruning of the schedule can result in a valid schedule. ETF is presented below in our notation and in a slightly different way to that given by Hwang *et al.* [1989]. Hwang *et al.* use a *General Delay* scheduling model where d can be expressed as the product of two functions: one with domain Δ and the other with domain $P \times P$. We relax this restriction.

In order to simplify the presentation of the algorithm we now define a little more notation.

Definition 4.3 First Event Time.
The *first event time* of a schedule $s \neq \{\}$, denoted $\mathcal{T}(s)$, is

$$\mathcal{T}(s) = \min_{(\gamma,p,t)\in s} t.$$

We also define $\mathcal{T}(\{\}) = \infty$. We note that in general, $|X(s, \Gamma, P, \mathcal{T}(s))|$ need not be 1.

Definition 4.4 First Event.
Given an instance $(P, \Lambda = (\Gamma, \Delta), f, d)$ of a scheduling model and a schedule $s \neq \{\}$, we let $\mathcal{F}$ be a function such that $\mathcal{F}(s)$ returns a single tuple chosen nondeterministically from $X(s, \Gamma, P, \mathcal{T}(s))$.

ETF works by stepping through time, keeping track of the tasks that are available for execution and the processors that are idle, and allocating tasks to processors whenever there are idle tasks and available processors. When there is a choice at any one moment about which task in the ready set is to be assigned to which processor in the idle set, task-processor pairings are made in the order in which they became possible, or rather would have become possible had the relevant processor not been busy. The algorithm is made slightly more complex because it does not move forward in steps of unit time. It only allocates tasks at time zero and at subsequent times when a processor has just finished executing a task, because only at these points will there be both idle tasks and idle processors.

In the algorithm below there are three schedules s, a, and r. s is the schedule being produced by ETF, and may be thought of as the output of the algorithm. a is a subset of s, and may be thought of as the set of tasks active at particular value of CM – the current moment. r is not really a schedule at all, at least not a valid schedule. It is used to record the times at which tasks will become available to processors, and may be thought of as the current set of available start times.

Tuples are present in r for every processor irrespective of whether or not they are idle, and as tasks are allocated to processors they are removed from the ready sets of all processors. The set of currently idle processors in the

above algorithm is I. It is initially set to all the processors in the instance of the scheduling model.

The outermost loop runs until all tasks have been allocated. The top inner while-loop allocates tasks to idle processors in the order in which such allocations became possible. It continues either until there are no processors left, or until the task would be allocated at a time after some processor finishes executing a task and becomes idle. When the top loop finishes, the current moment moves on to the next time a processor finishes executing a task, and the bottom inner while loop starts considering all the tasks that have just finished executing.

If any of these tasks is the last to finish of the predecessors of some task, then that task is added to every processor's ready list. The time at which it will become available to each processor is set to the time at which the last of the results of its predecessor tasks will become known to that processor. Note that because of non-uniform communication delays the last predecessor known to a processor is not necessarily the last one to complete execution.

The ETF algorithm is, in terms of our notation, as follows.

Algorithm 4.4 ETF

1 let $(P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a *General Delay* scheduling model.

2 for all $\gamma \in \Gamma$ *let* $Q(\gamma) = |\{\delta \in \Gamma : (\delta, \gamma) \in \Delta\}|$

3 let $s = a = \{\}$;

4 let $I = P$;

5 let $r = \{(p, \gamma, 0) : p \in P, \gamma \in \Gamma, Q(\gamma) = 0\}$

6 let $CM = 0$;

7 While $|s| < |\Gamma|$

do

8 While $\mathcal{T}(X(r, \Gamma, I, Z)) \leq \mathcal{T}(a)$

do

9 let $(\gamma, p, t) = \mathcal{F}(X(r, \Gamma, I, Z))$;

10 let $e = \max\{CM, t\}$;

11 let $r = r - X(r, \gamma, P, Z)$;

12 let $s = s \cup \{(\gamma, p, e)\}$;

13 let $a = a \cup \{(\gamma, p, e + f(\gamma))\}$;

14 let $I = I - \{p\}$;

enddo

15 let $CM = \mathcal{T}(a)$;

16 While $\mathcal{T}(a) = CM$

do

17 let $(\gamma, p, t) = \mathcal{F}(a)$;

18 let $I = I \cup \{p\}$;

19 for each $(\gamma, \delta) \in \Delta$

do

20 let $Q(\delta) = Q(\delta) - 1$;

21 if $Q(\delta) = 0$

do

22 for each $q \in P$

do

23 let $t_{max} = \max_{(\beta, q', t') \in X(s, \{\alpha : (\alpha, \delta) \in \Delta\}, P, Z_0^+)} (t' + f(\beta) + d((\beta, \delta), q', q))$;

24 let $r = r \cup \{(\delta, q, t_{max})\}$;

enddo

enddo

25 let $a = a - \{(\gamma, p, t)\}$;

```
        enddo
    enddo
enddo
end
```

4.3.2 The ETF bound

Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a *General Delay* scheduling model. where $|P| = n$. Let $B = (P, \Lambda = (\Gamma, \Delta), f, d')$ be a corresponding instance of a *No Delay* scheduling model.

Let s^A_{ETF} be a schedule produced by Algorithm 4.4 for A. The performance bound on $\mathcal{M}(s^A_{ETF})$ is given in terms of the makespan of an optimal schedule, say s^B_{opt} of B. Hwang *et al.* prove a bound on $\mathcal{M}(s^A_{ETF})$ which can be generalised to the following.

$$\mathcal{M}(s^A_{ETF}) \leq (2 - 1/n)\mathcal{M}(s^B_{opt}) + c$$

where

$$c = \max_{\{(\gamma_1,\gamma_2),(\gamma_2,\gamma_3),\ldots,(\gamma_{r-1},\gamma_r)\}\subseteq\Delta} \sum_{i=1}^{r-1} \max_{p,q\in P} d((\gamma_i, \gamma_{i+1}), p, q).$$

4.4 Comparing ETF and PNY

As a result of the differences between the underlying models, there are difficulties in comparing PNY with ETF directly. Instead we compare schedules produced by Algorithm 4.3 that have been remapped by Algorithm 3.2 and then round robin rescheduled according to Definition 4.2. Furthermore we restrict our attention to UET *Fixed Delay* scheduling models.

4.4.1 A Comparison of Bounds

In the following analysis we will be referring to related instances of *four* scheduling models.

- Fl is an instance of a *Fixed Delay* scheduling model with l processors.
- Nl is an instance of a *No Delay* scheduling model with l processors.
- Fa is an instance of a *Fixed Delay* scheduling model with a processors.
- Na is an instance *No Delay* scheduling model with a processors.

Let P be a set of l processors. Let $Fl = (P, \Lambda = (\Gamma, \Delta), f, d_{Fl})$ be an instance of a *Fixed Delay* UET scheduling model where $|P| = l$ and the range of d is $\{0, \tau\}$. Let $Nl = (P, \Lambda = (\Gamma, \Delta), f, d_{Nl})$ be a corresponding instance of a *No Delay* scheduling model. Let $a \in Z_1^+$ be a number of processors such that $a \leq \mathcal{W}(\Lambda)\lceil(\tau + 1/2)\rceil$. Let $p_1, \ldots, p_l$ be an enumeration of P. Let $P' = \{p_1, \ldots, p_a\}$. Let d_{Fa} be the restriction of d_{Fl} to the domain $\Delta \times P' \times P'$. Let d_{Na} be the restriction of d_{Nl} to the domain $\Delta \times P' \times P'$. Let $Fa = (P', \Lambda, f, d_{Fa})$. Let $Na = (P', \Lambda, f, d_{Na})$.

Let s_{opt}^{Fl} be an optimal schedule of Fl. Let s_{opt}^{Nl} be an optimal schedule of Nl. Let s_{opt}^{Na} be an optimal schedule of Na.

Let s_{PNY}^{Fl} be a schedule generated by Algorithm 4.3 for Fl. Let s' be the result of applying Algorithm 3.2 to s_{PNY}^{Fl}. Let s_{PNY}^{Fa} be the round-robin remapping of s' to a processors. By Corollary 4.2

$$\mathcal{M}(s_{PNY}^{Fa}) \leq 2\lceil\frac{\mathcal{W}(\Lambda)\lceil(\tau + 1)/2\rceil}{a}\rceil\mathcal{M}(s_{opt}^{Fl})$$

Let s_{ETF}^{Fa} be a schedule produced by ETF for Fa. In the special case of the UET *Fixed Delay* scheduling model the bound of Section 4.3.2 can be expressed as

$$\mathcal{M}(s_{ETF}^{Fa}) \leq (2 - 1/a)\mathcal{M}(s_{opt}^{Na}) + \tau\mathcal{H}(\Lambda).$$

These bounds are not easily comparable since the PNY bound is expressed in terms of $\mathcal{M}(s_{opt}^{Fl})$ and the ETF bound in terms of $\mathcal{M}(s_{opt}^{Na})$. We now give a new bound on ETF which is in terms of $\mathcal{M}(s_{opt}^{Fl})$.

As an intermediate result Hwang *et al.* [1987], when restated for the restricted case of *Fixed Delay* UET scheduling models, show the following

$$\sum_{t=1}^{\mathcal{M}(s_{ETF}^{Fa})} a - |\mathcal{A}(s,t)| \leq (a-1)\mathcal{H}(\Lambda) + a\tau(\mathcal{H}(\Lambda)-1).$$

Note

$$\mathcal{M}(s_{opt}^{Fa}) \geq \mathcal{M}(s_{opt}^{Na}).$$

Note also

$$\mathcal{M}_{opt}^{Nl} \geq \mathcal{H}(\Lambda)$$

since in any schedule the tasks of a path of length $\mathcal{H}(\Lambda)$ must be processed in sequence, and finally note

$$\mathcal{W}(\Lambda)\mathcal{H}(\Lambda) \geq l.$$

Now, since in s_{ETF}^{Fa} a processor is either processing one of the l tasks in Γ, or idle.

$$\begin{aligned}
\mathcal{M}(s_{ETF}^{Fa}) &= 1/a(l + \sum_{t=1}^{\mathcal{M}(s_{ETF}^{Fa})} a - |\mathcal{A}(s,t)|) \\
&\leq 1/a(\mathcal{W}(\Lambda)\mathcal{H}(\Lambda) + (a-1)\mathcal{H}(\Lambda) + a\tau(\mathcal{H}(\Lambda)-1)) \\
&\leq 1/a(\mathcal{W}(\Lambda) + a(\tau+1) - 1)\mathcal{M}(s_{opt}^{Fl}) - \tau \\
&\leq (\frac{\mathcal{W}(\Lambda)}{a} + (\tau+1))\mathcal{M}(s_{opt}^{Fl})
\end{aligned}$$

In comparing the bounds it must be noted that ETF is a more general-purpose algorithm than PNY. ETF might not be expected to perform as well as PNY on the special case of *Fixed Delay* UET scheduling models to which PNY uniquely applies.

4.4.2 Exemplar Schedules

We can consider the use of ETF and PNY on different scheduling models. Again we restrict our consideration to the set of scheduling models to which they may both be applied. We consider two such models. The first is one on which PNY performs spectacularly badly. The second is one on which ETF performs almost as spectacularly badly.

The Lumpy dag

Let τ be a positive integer. Let us define task graph $\Lambda = (\Gamma, \Delta)$ and a set of processors P as follows.

$$\begin{aligned}
\Gamma &= \{\gamma_x^y | 1 \leq x \leq \tau, 1 \leq y \leq 2\tau + 1\} \cup \{\gamma_0^0\} \\
\Delta &= \{(\gamma_x^y, \gamma_x^{y+1}) | 1 \leq x \leq \tau, 1 \leq y < \tau\} \cup \\
&\quad \{(\gamma_x^y, \gamma_x^a) | 1 \leq x \leq \tau, 1 \leq y \leq \tau, \tau + 1 \leq a \leq 2\tau + 1\} \cup \\
&\quad \{(\gamma_x^y, \gamma_0^0) | 1 \leq x \leq \tau, \tau + 1 \leq y \leq 2\tau + 1\} \\
P &= \{p_1, p_2, \ldots, p_{|\Gamma|}\}
\end{aligned}$$

Figure 4–3 shows Λ for $\tau = 3$. Vertices are labelled on the figure simply with their two indices. Let $A = (P, \Lambda, f, d)$ be an instance of a *Fixed Delay* UET scheduling model where the range of d is $\{0, \tau\}$. Let $P' = \{p_1, p_2, \ldots, p_\tau\}$. Let d' be the restriction of d to the domain P'. Let $B = (P', \Lambda, f, d')$.

Let us consider some schedule s_{ETF}^B of B generated by Algorithm 4.4. First note that only tasks γ_x^1 $1 \leq x \leq \tau$ have no predecessors so these form the initial ready set, and without loss of generality task γ_x^1 is assigned to processor p_x to execute at time 0. When each of these tasks finishes, task γ_x^2 is assigned to processor p_x since it becomes ready to execute on that processor (but not on any other processor for which a communication delay would be required). The allocation continues until each task γ_x^τ $1 \leq x \leq \tau$ completes its execution on

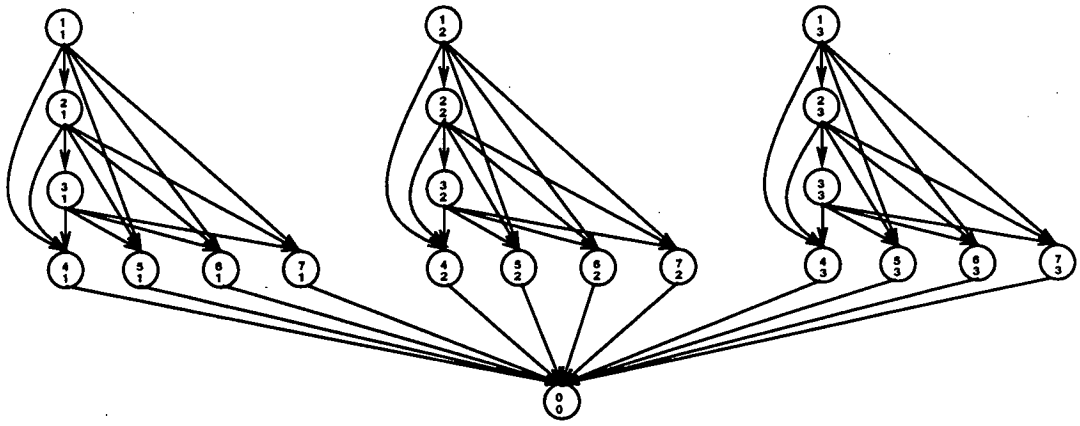


Figure 4–3: The Lumpy dag for $\tau = 3$

processor p_x at time τ. At this time tasks γ_x^y $1 \leq x \leq \tau, \tau+1 \leq y \leq 2\tau+2$ become ready for execution on their respective processors p_x. ETF is free to schedule these tasks on their respective processors in any order and since their connections are symmetric, without loss of generality we may assume that it schedules tasks γ_x^y $1 \leq x \leq \tau, \tau+1 \leq y \leq 2\tau+1$ at time $y-1$. This means that each processor p_x finishes computing task $\gamma_x^{2\tau+1}$ at time $2\tau+1$, and then finally task γ_0^0 becomes available to all processors at time $3\tau+1$ and some processor, say p_1, executes it and finishes at time $3\tau+2$. That is

$$\mathcal{M}(s_{ETF}^B) = 3\tau + 2.$$

Now consider applying Algorithm 4.1 to Λ. The e-value of each task $\gamma_x^y : 1 \leq x \leq \tau, 1 \leq y \leq \tau$ is $xy-1$, since they have $y-1 < \tau$ predecessors. The e-value of tasks $\gamma_x^y : 1 \leq x \leq \tau, \tau+1 \leq y \leq 2\tau+2$ is τ since they all have τ predecessors. Finally the e-value of task γ_0^0 is $2\tau+1$ since its τ highest in e-value predecessors (indeed its $\tau^2+\tau$ immediate predecessors) all have e-value τ, and therefore we must add $\tau+1$ to the e-value of one of these predecessors.

Let us consider some schedule s_{PNY}^A of A generated by Algorithm 4.3. In this schedule the last task to be executed is γ_0^0 at time $2e(\gamma_0^0)$ so

$$\mathcal{M}(s_{PNY}^A) = 4\tau + 3.$$

Unfortunately such a schedule would use $\tau^2 + 2\tau + 1$ processors, where at least $(\tau + 1)\tau$ processors were active for $\tau + 1$ time units. Let us consider some schedule s^B_{PNY} of B generated by rescheduling s^A_{PNY}. No matter how clever an algorithm was used to reschedule to the τ processors this means that at least one processor would be executing for at least $(\tau + 1)^2$ time units. Thus

$$\mathcal{M}(s^B_{PNY}) \geq (\tau + 1)^2.$$

The Inverse Binary Tree

Our second example is the inverse binary tree of Jung *et al.* [1989]. Given an integer value τ we construct our instance $C = (P, \Lambda = (\Gamma, \Delta), f, d)$ of a *Fixed Delay* UET scheduling model where

$$\begin{aligned} \Gamma &= \{\gamma_i : 1 \leq i \leq 2^\tau - 1\}. \\ \Delta &= \{(\gamma_i, \gamma_{2i}), (\gamma_i, \gamma_{2i+1}) : 1 \leq i < |\Gamma|/2\}. \\ P &= \{p_1, p_2, ..., p_{2^\tau}\}. \end{aligned}$$

Consider applying Algorithm 4.1 to Λ. The e-value of each task $\gamma_i : 1 \leq i \leq 2^\tau$ is $\lceil \log i \rceil$ since all tasks have that number of predecessors which is fewer than τ. Let us consider some schedule s^C_{PNY} of C generated by Algorithm 4.3. Thus

$$\mathcal{M}(s^C_{PNY}) = 2(\tau - 1) + 1 = 2\tau - 1.$$

Let us consider some schedule s^C_{ETF} of C generated by Algorithm 4.4. Some processor, say p_1, is allocated task γ_1. After executing task γ_1 the two children of that task become available for execution on p_1 at time 1, and on all other processors at time $\tau + 1$. Processor p_1 continues executing tasks of the tree in a breadth first traversal of the tree. Task γ_i is executed at time $i - 1$ and for every task executed, two tasks are added to the set of tasks which are available

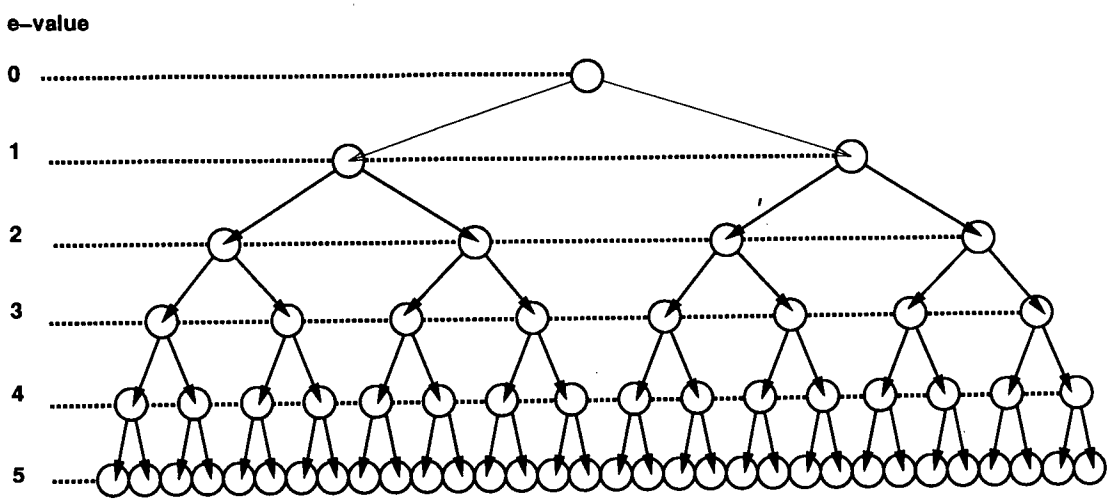


Figure 4–4: The Inverse Binary Tree for $\tau = 6$ and its e Values

for execution on processor p_i. Eventually, at time 2τ, task $\gamma_{2\tau}$, which has been available for execution on processor p_i since time τ, has not been executed and becomes available for execution on all other processors. Note that from this time step on, processor p_1 cannot execute tasks as fast as its execution of their parents make them available to it. For each 2τ tasks available to it, it executes τ. Together these make 2τ successors available to it, of which it can execute τ, so at each subsequent level of the tree it executes τ tasks. Since there are at least $\tau - \lfloor \log(2\tau) \rfloor$ subsequent layers to the tree after the first 2τ tasks,

$$\mathcal{M}(s^C_{ETF}) \geq 2\tau + \tau(\tau - \lfloor \log(2\tau) \rfloor)$$

and thus

$$\mathcal{M}(s^C_{ETF}) \geq \tau(\tau + 1 - \log \tau).$$

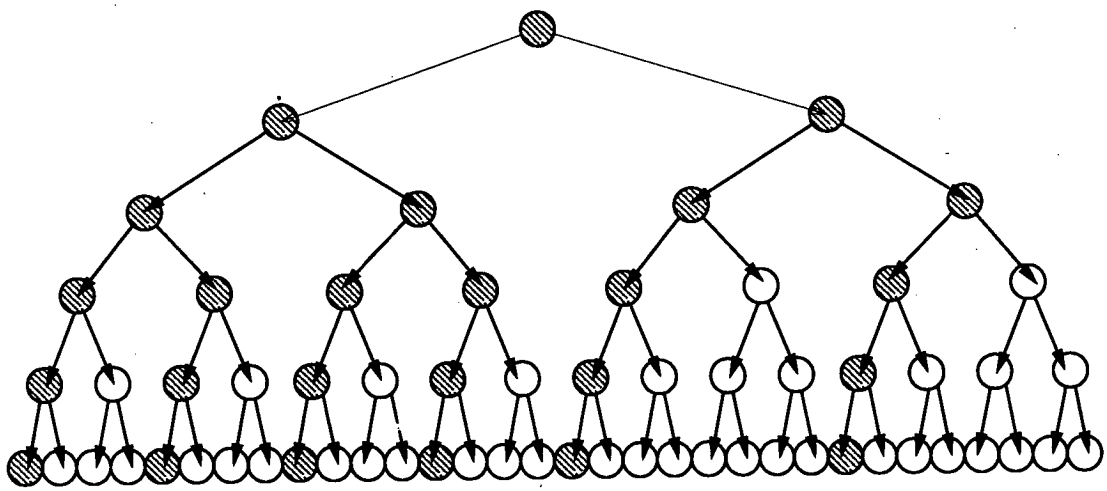


Figure 4–5: ETF Schedule of the Inverse Binary Tree for $\tau = 6$

Chapter 5

Process-Based Models

5.1 An Overview of Process Based Models

The scheduling models of Chapters 2, 3 and 4 contrast with another approach to modelling computations on multicomputers whereby modules (which in this context we refer to as processes) are arranged in undirected graphs, and edges between processes represent two-way communication rather than precedence and one-way communication. These models are often used for modelling computations at a coarse granularity where modules are persistent processes which exist for the duration of a computation, and there are stable communication patterns between them.

In many cases, there is an equivalence between a process based model of computation and a probabilistic, and possibly cyclic, graph of task dependencies and branching probabilities. The former can be derived from the latter by applying a Markov chain analysis. An example of the approach is to be found in Chu *et al.* [1984].

In this chapter we describe a pair of models, the first by Stone [1977a], and the second by Bokhari [1981b]. The two models differ in the way in which communication is handled and both are limited in application. In Chapter 6 we consider the complexity of mapping in a version of Bokhari's model. In Chapter 7 we describe various approaches to extending the applicability of such models, and for integrating them with scheduling models described earlier.

It is worth bearing in mind that a number of the models discussed in this and later chapters were presented in the context of distributed systems rather than multicomputer systems. We include them in our review, firstly because there is a grey area between the two types of system and, secondly, because several papers can be viewed as attempts to adapt these research results to mapping problems for multicomputers. Unfortunately, there are significant differences in

Feature	Distributed Systems	Multicomputers
Work Profile	Multiple Job	Single Job
Processor Type	Heterogenous	Homogenous
Fault Tolerance	Important	Ignored
Optimisation	Maximise Throughput by Minimising System Resource Consumption	Minimise Time to Completion
Ratio of Message Latency to Instruction Cycle Time	Relatively High	Relatively Low
Individual Job Execution	Can be Required to be Sequential	Parallel

Table 5–1: Relevant Differences Between Models of Distributed Systems and Models of Multicomputer Systems

the abstract models of computation being used by researchers in the two fields; these are summarised in Table 5–1. We are not saying that all researchers make these assumptions, rather we are conveying an impression of the differences in emphasis between the two fields.

5.2 Definitions

Below we develop a notation based on graphs (rather than directed graphs) which is used in this chapter and in Chapter 6.

Definition 5.1 Graph.
A *graph* (rather than a *directed* graph) say, $G = (V, E)$, is an ordered pair comprising a set, V of *vertices* and a set, E, of *edges*. An edge in the set E is an unordered pair of distinct vertices from the set V.

Definition 5.2 Degree of Vertex and of Graph.

Given a graph $G = (V, E)$, the *degree* of a vertex $v \in V$ is the cardinality of the set containing all edges $\{u, v\} \in E$ where $u \in V$. The *degree* of G is the degree of an vertex in V of maximal degree.

Definition 5.3 Path.

A *path* in a graph $G = (V, E)$ between a pair of vertices $\{u, v\}$ is a sequence of vertices $\{\theta_1, \theta_2, \ldots, \theta_k\} \subseteq V - \{u, v\}$, $k \geq 0$, such that

$$\{u, \theta_1\}, \{\theta_1, \theta_2\}, \ldots, \{\theta_{k-1}, \theta_k\}, \{\theta_k, v\} \in E .$$

Definition 5.4 Path Length, Shortest Path Length.

Given a graph $G = (V, E)$, we define the *length* of such a path $\{\theta_1, \theta_2, \ldots, \theta_k\} \subseteq V - \{u, v\} -$ to be $k - 2$. We also define a function $SH_G(\{u, v\})$ to be the length of the (possibly non-unique) shortest path between u and v in G. Note that, our definition of *path length* is slightly different from the usual graph theoretic definition. So, we have $SH_G(\{u, v\}) = 0$ if and only if $\{u, v\} \in E$.

Definition 5.5 Shortest Path.

Given a graph $G = (V, E)$, and a pair of vertices, u and v in V, we define a function PA_G such that if there is a unique *shortest path* between u and v, $PA_G(\{u, v\})$ returns that path. Otherwise $PA_G(\{u, v\})$ is undefined.

Definition 5.6 Routing Load.

Given a graph $G = (V, E)$, and a pair of vertices, u and v in V, we define $RO_G(\{x, y\}, v)$ to be a function which indicates if a vertex v is in the unique shortest path between x and y. That is it returns 1 if $v \in PA_G(\{x, y\})$, or 0 otherwise.

Note that, by our definition, if a shortest path exists then

$$\sum_{v \in V} RO_G(\{x, y\}, v) = SH_G(\{x, y\})$$

and $\sum_{x,y \in V} RO_G(\{x,y\},v)$ is equal to the number of unique shortest paths on which the vertex v is located.

Definition 5.7 Process Based Model.
An instance of a *process based* model is a 4-tuple (P,G,f,c). P is a set of n processors, $G=(V,E)$ is a graph, where V is a set of vertices corresponding to processes, and E is a set of edges corresponding to communication between processes. $f : V \times P \rightarrow Z_0^+$ is a function such that $f(v,p)$ returns the cost of computing task $v \in V$ on processor $p \in P$; $c : E \times P \times P \rightarrow Z_0^+$ is a function returning the cost associated with communication between processes if they are mapped to different processors.

Definition 5.8 Process Mapping Function.
A *process mapping function* for an instance $(P,G=(V,E),f,c)$ of a process based model is a function

$$g : V \rightarrow P$$

which maps each process to a processor. We define a corresponding function

$$\bar{g} : P \rightarrow 2^V$$

which returns the set of processes mapped to a given processor by mapping function g.

Definition 5.9 Computation Cost of a Mapping.
Given an instance $(P,G=(V,E),f,c)$ of a process based model, the global *cost* of computation, $\mathcal{U}(g)$ associated with mapping function g, is given by:

$$\mathcal{U}(g) = \sum_{v \in V} f(v,g(v)).$$

Definition 5.10 Communication Cost of a Mapping.
Given an instance $(P,G=(V,E),f,c)$ of a process based model, the global *cost*

of communication, $\mathcal{V}(g)$ associated with mapping function g, is given by:

$$\mathcal{V}(g) = \sum_{\{v,w\} \in E, g(v) \neq g(w)} c(\{v,w\}, g(v), g(w)).$$

Definition 5.11 Cost of a Mapping.
Given an instance $(P, G = (V, E), f, c)$ of a process based model the global *cost* $\mathcal{C}(g)$ of a mapping is given by:

$$\mathcal{C}(g) = \mathcal{U}(g) + \mathcal{V}(g).$$

5.3 Stone's 1977 Model

The basis of Stone's model [Stone 1977a] is that there is a computation cost associated with executing each process and a communication cost associated with sending each inter-process message. The total cost of a mapping is the sum of all the computation costs and communication costs. For reasons discussed later, Stone's model is formulated for non-identical processors, where the computation cost of executing a process depends on the processor upon which it is executed.

Decision Problem 5.1 *Given an instance A of a process based model and an integer k: does there exist a process mapping function, g, for A such that $\mathcal{C}(g) < k$?*

5.3.1 Understanding Stone's model

We can rationalise Stone's model in terms of our framework as outlined in Table 5–2. Here we are associating $\mathcal{C}(g)$ with the time to completion of the

$T_{Calc}(p)$		$\sum_{v \in \bar{g}(p)} f(v,p)$
T_{Comm}		0
T_{House}		0
T_{Idle}	T_{Wait_D}	$\mathcal{V}(g)$
	$T_{Wait_S} + T_{Fin}$	$(n-1)\mathcal{C}(g)$

Table 5–2: A Process Based Model in Terms of our Framework

program: as Bokhari points out [Bokhari 1981b] there is nothing in the model to enforce this; $\mathcal{C}(g)$ could be measured, for example, in dollars. In our rationalisation $T_{Calc}(p)$ is simply the execution time of the processes that have been assigned to processor p. The total time processors spend waiting for a message to be delivered, T_{Wait_D} is simply the total communication cost. All other time is spent in T_{Wait_S} or T_{Fin}.

The rationalisation stems from the fact that Stone's work considers only the case of sequentially executing processes. That is at any one time exactly one process is executing on one of n processors, and the other processors are idle for the duration of its execution. The cost of communication between processes corresponds to a delay between a process terminating and another process starting. The *sum* of the costs corresponds to the time to completion of the program since the costs are incurred sequentially. Stone clearly states that he is dealing with these serial programs, but it is an issue that could easily be overlooked when considering other papers referencing his model.

As a result of the sequential nature of execution in this cost model, in the case of machines with identical processors such as most multicomputers, it will always be optimal to execute tasks on a single processor. To do otherwise will never reduce computation cost, and can only ever incur extra communications cost.

5.3.2 Results and Algorithms

There are a number of results for this model which relate to two-processor systems. These stem from work by Stone [1977a, 1978] based upon the use of network flow diagrams. Stone shows an optimal algorithm for a process based model with $n = 2$. An extension of Stone's work to three processors was performed by Stone himself [1977b] and Bokhari [1981b] cites an unpublished result of Gursky that the four or more processor versions are NP complete.

Fernández-Baca [1989] shows that Decision Problem 5.1 is NP complete even if all of the following restrictions hold:

- the range of f is $\{0\}$,
- the range of c is $\{1\}$,
- $n = 3$,
- G is both planar and bipartite.

Furthermore, Fernández-Baca [1989] showed that there can exist no ϵ approximation algorithms for the problem constrained in the above fashion and no exact local search algorithm that takes polynomial time per iteration. This rather negative result is balanced by the claim by Sinclair [1987] that the *average* time-complexity is manageable in this problem for a version of exact branch and bound search.

There are also positive results for special classes of process graphs. Bokhari [1981b] shows an algorithm that, if the process graph is a tree, is guaranteed to find an exact solution in $O(ln^2)$ time. Towsley [1986] applies a dynamic programming approach to the problem with series-parallel graphs, and shows an algorithm with time complexity $O(ln^3)$. Fernández-Baca[1989] extended the results to other tree-like graphs. Rao *et al.* [1979] consider the case of Stone's

original model but where one of the processors is constrained in memory, and show two techniques which can reduce the complexity of the problem in some cases but not in general. Gusfield [1983] solves the problem with a parametric computing technique, for the costs of mapping processes to the two processors varying as a function of two independent parameters.

5.3.3 Adding Constraints to Stone's Model

There have been a number of direct extensions to Stone's work whereby the underlying model has been extended to allow constraints upon the solution, for example memory resource constraints. Given the NP completeness of the model without constraints, heuristic approaches to the problem of determining optimal mappings have been proposed for these extended models. Chu *et al.* [1980], in an early review of the field, consider integer programming approaches in the presence of constraints. Gylys and Edwards [1976] describe process clustering algorithms which satisfy constraints. Ma *et al.* [1982] use a branch and bound technique to solve a model which includes constraints, including a *redundancy* constraint: certain processes must be allocated to more than one processor.

Other authors have used constraints and costs to attempt to encourage the potential of parallelism between processes in the resulting mapping. Lo [1988] considers an alternative to our process based model where if task v is mapped to processor p, $c(v,p)$ is dependent upon the elements of $\bar{g}(p)$: that is, there is a cost associated with interference between processes. Houstis [1990] (in the context of real-time systems) adds an explicit parallel processing constraint to the model: if two processes *can* be executed in parallel then they *must* be executed on different processors.

5.4 Bokhari's 1981 Model

In cases where computation and communication costs are not incurred sequentially (ie. in parallel rather than serial programs) Stone's model is not directly applicable since communication costs and computation costs are no longer strictly additive. As an alternative, we might consider algorithms which attempt to find minimum communication cost mappings irrespective of computation costs. Models of this nature often include considerations of some underlying processor architecture. The problem becomes that of mapping an undirected graph of processes into an undirected graph of processors so that one process is mapped to each processor and so as to minimise the communication overhead.

To return to our example graph, we now consider the graph to be undirected, and the edges to indicate volumes of communications between processes. We consider the problem of mapping such processes to a processor graph which, for the purposes of our example, we shall assume is a seven processor ring. Clearly it is not possible to place all communicating processes on adjacent processors: each processor has exactly two adjacent processors whereas process D communicates with three processes. The mapping we show in Figure 5–1 is optimal in the sense that no interprocess communication is extended by more than one interprocessor link – that is it has a dilation of 1.

To describe these approaches more thoroughly we use a new type of model which we refer to as a *graph matching* model. For a given graph matching model we can construct a corresponding process based model.

Definition 5.12 Graph Matching Model.
A *graph matching* model consists of a 4-tuple, say , (Q, G, e, c') where $Q = (P, L)$ is a graph of n processors (the edges correspond to interprocessor links); $G = (V, E)$ is a process graph where V is a set of n processes (ie. there are as many

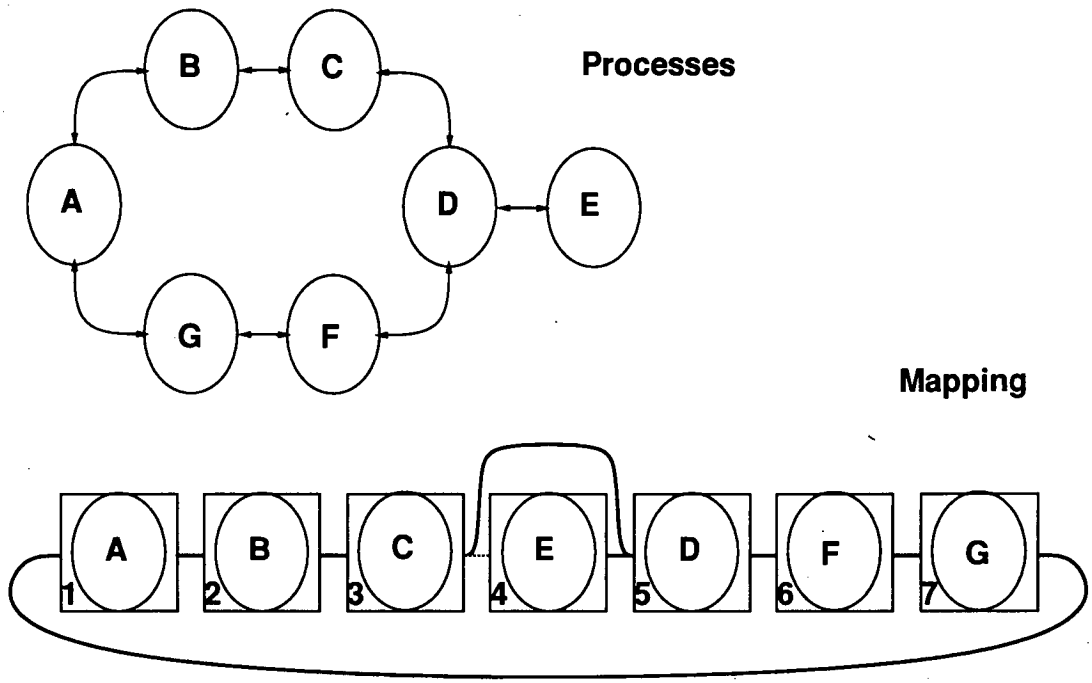


Figure 5–1: An Example of Bokhari's Mapping Problem and its Solution

processes as there are processors) and an edge $\{u, v\} \in E$ implies processes u and v communicate with each other. $c' : E \rightarrow Z_0^+$ is a function such that $c'(\{u, v\})$ returns an integer corresponding to the amount of communication that u performs with v (bidirectionally) during their execution and e is an integer execution time which is the same for each process.

Definition 5.13 Process Based Model Corresponding to Graph Matching Model. Given an instance $(Q = (P, L), G = (V, E), e, c')$ of a Graph Matching Model we can construct a corresponding instance (P, G, f, c) of a process based model where for all $\{u, v\} \in E$, for all $p, q \in P$, $p \neq q$,

$$c(\{u, v\}, p, q) = SH_Q(\{p, q\}) \times c'(\{u, v\})$$

and for all $u \in V$, $f(u) = e$.

We have the following decision problem for mapping in a graph matching model. Note the mapping function is restricted to be a bijection.

Decision Problem 5.2 *Let k be an integer and $A = (Q, G, e, c')$ be an instance of a graph matching model. Let B be the instance of a process based model which corresponds to A. Does there exist a one-to-one mapping function g for B such that $C(g) \leq k$?*

Decision Problem 5.2 corresponds to the question: can I allocate processes to processors in a given processor topology so as to minimise dilation? Bokhari [1981a] points out that in the case where $k = 0$, Decision Problem 5.2 is a notational variant of the graph isomorphism problem. As Bokhari noted the complexity status of graph isomorphism is an open problem (see Garey and Johnson [1979])[1]. It is possible that graph isomorphism affords a polynomial time algorithm in the types of regular graphs of bounded degree that were under consideration by Bokhari.

5.4.1 Bokhari's Model in Terms of the Framework

At first sight it is not clear how a model that fails to take into consideration computation overheads can relate well to the mapping problem for multicomputers. However, as Fox observed in a survey of parallel applications [Fox 1989], 76% of real parallel programs being run were what he considered to be synchronous or loosely synchronous.

These programs would be executed on a multicomputer by regular data decomposition with one process per processor; roughly equal amounts of computation per process; and periodic interprocess communications. Moreover, as a result of global or pairwise synchronisations between processes the computation would execute in a roughly lock-step manner.

---

[1]The incorrect notion that graph isomorphism is NP complete is endemic in literature on the mapping problem; see, for example, Krämer and Mühlenbein [1989] and Pountain [1989].

$T_{Calc}(p)$		e
T_{Comm}		0
T_{House}		0
T_{Idle}	T_{Wait_D}	$C(g)$
	T_{Wait_S}	0

Table 5–3: A Graph Matching Model in Terms of our Framework

Given the inherent load-balance of such problems the problem of mapping such computations to a multicomputer is simply that of allocating processes to processors to minimise communications overhead, and it is likely that the *dilation* of the process graph will adequately model the loss in performance due to non-local communication. We discuss this issue again in Chapter 6 below.

It is instructive to consider a graph matching model in terms of our framework. If we assume that all processes start waiting for communications at exactly the time they are sent, then the cost of the message, which is dependent upon interprocessor distance, makes sense as a communication latency, which that process must wait upon, and therefore shows up in T_{Idle} in our framework as outlined in Table 5–3.

Another possible way of reconciling the cost of communication in terms of our framework is to consider it to be contributing to T_{Route}. Unfortunately, this does not bear up to close scrutiny since in Bokhari's model it is the extension of individual messages that is being considered harmful whereas through-routing costs are additive on a *per-processor* basis. The function c does not apportion through-routing computation to a particular processor along the shortest route(s), and so the relationship between the through-routing costs and the time to completion of the program is undefinable.

Given the data-parallel rationalisation above, it is perhaps not surprising that

formulations such as Bokhari's are used in the analysis of algorithms for SIMD architectures. In these machines the mapping problem is more constrained and has been the subject of significant analysis. An introduction and a significant bibliography is to be found in [Johnsson 1990] and also in [Leighton 1992]. The multicomputer programmer may find such approaches useful in analysing the performance of regular data parallel computations, but should be wary of the validity of assuming a lock-step computation.

Chapter 6

Message Contention

6.1 Contention in the Multicomputer

In many multicomputer applications, in particular in those where the structure of the computation and the structure of the multicomputer are not similar, it may not be appropriate to treat each communication as if it were sent down an independent piece of hardware, rather it may be assumed that communications sometimes interact with one another to delay each other. This interaction we shall refer to as *message contention*. In this chapter we briefly review the literature on the subject, and consider the complexity of two problems which, in very different ways, may be thought of as mapping problems where message contention is taken into account.

First we review the literature which concerns contention in the multicomputer rather than the impact of contention on mapping. There are different issues in packet switched systems and circuit switched or wormhole systems.

6.1.1 Packet Switched Systems

There is a substantial literature on contention in packet switched systems which is eloquently summarised in [Leighton 1992] and [Johnsson 1990]. The focus is the *packet routing problem* where a number of processors arranged in some topology are required to send at most one message, each to a distinct destination, and the objective is to minimise the amount of time that this operation takes, or the amount of queuing resource that is required.

As an alternative to the above, it is possible to consider the routing techniques where the route that messages take is chosen at random, in the sense that messages are sent to a random processor before being sent to their final destination. See, for example [Valiant and Brebner 1981], [Valiant 1982] and Upfal [1984]. Now, the expected time to completion of a set of messages can

be shown to be probabilistically bounded and independent of the message set. Both best case performance and worst case performance are expected to be around half the average case performance for deterministic strategies, and locality in the processor network is lost. To most multicomputer programmers the loss in performance may appear Quixotic, but this approach may prove useful in mapping, for example in the models outlined in Section 2.3. In practice, however, random routing is not implemented in hardware in commercially available multicomputers' routing systems; the overheads of random routing implemented in software (which we would associate with T_{Route}) could often outweigh any advantage.

6.1.2 More Complex Interconnection Systems

Most of the above is rather academic because the routing strategy and the hardware topology are rarely under the control of the multicomputer programmer, and the majority of existing multicomputer architectures based on hypercubes, for example use the simple deterministic e-cube routing strategy, which is known to have very poor worst-case performance [Dally 1990] . In addition, the majority of currently available multicomputers have wormhole or circuit switched interconnects which behave very differently from the simple packet switched systems described above. Moreover some newer systems have inherent flow control mechanisms which make analysis of contention extremely difficult.

Queueing theory based approaches have been successful to some degree in analysing the performance of more complex interconnects. Good examples are Dally's analysis of the k-ary n-cube [Dally 1990b], Chittor and Enbody's work on the iPSC860 [Chittor and Enbody 1992] and Scott *et al.* [1992] on the proposed IEEE standard Scalable Coherent Interconnect [IEEE 1991].

6.2 Contention in a Process Based Model

In this section we consider the complexity of a *process based* mapping problem in the presence of contention. Recall the *graph matching* model outlined in Chapter 5. The formulation associates *cost* with the total number of message-link transfers that are implied by a mapping. Clearly in these models, which do not incorporate temporal aspects, we are unable to consider the detailed aspects of how messages interact, but since the total number of links in the processor graph is fixed, our cost function may be thought of as measuring the total level of message traffic contending for interprocessor message links.

There are some obvious extensions to the graph matching model. First it is possible to consider the cost of communication to be the maximum of the sum of the message link utilisation levels along the path taken by any message. This approach is taken by Lee and Aggarwal [1987] and Berman and Snyder [1987]. The second extension is to consider the problem when there are more processes than processors. Berman and Snyder [1987] extend the analysis to consider multiprocessing, having contracted the process graph to have the same number of processors as the processor graph.

Alternatively, approaches to modelling contention in process based models are often couched directly in terms of the corresponding probabilistic model (recall Section 5.1), since it can be integrated with probabilistic queuing theory models, and then solved with Markov chain analysis in the same way as the process based model is derived. Examples of these approaches are given by Houstis [1990] and Cvetanovic [1987] who consider the communications medium as a saturable bus, whose performance for any communication is determined by its rate of utilisation, and Kapelnikov *et al.* [1989] who model multicomputer networks. The models are complicated and are usually presented

as modelling methodologies rather than being proposed as objective functions in mapping problems.

Below we consider a graph matching model, from the point of view of the problem of *building a multicomputer whose structure matches the computation*. As stated in Chapter 5, Bokhari formulated a graph matching model and showed the problem reduces to graph isomorphism of which the complexity is open. In this section we consider a graph matching based mapping problem whereby the *processor graph* is the variable to be optimized. The aim is to build a multicomputer which minimises total message traffic for a given computation modelled as a process graph.

6.3 The Multicomputer Configuration Problem

Before we state our decision problem, we will first establish two lemmas. Note that we are using the definitions of Chapter 5 for the *routing load* and the *shortest path* in an (undirected) graph.

Lemma 6.1 *Let $G = (V, E)$ be a graph. Let $V' \subseteq V$ be such that $|V'| = a$. Let $E' \subseteq \{\{t_i, t_j\} : t_i, t_j \in V'\}$ Then, the following inequality holds.*

$$\sum_{v \in V'} \sum_{e \in E'} RO_G(e, v) \leq \frac{a^3 - 3a^2 + 2a}{2}.$$

Proof

We note that adding an edge to E' cannot decrease the sum. Thus the left-hand side of the inequality will be maximised when $G' = (V', E')$ is a complete graph. In this case each vertex $v \in V'$ will have edges to at most $(a-1)$ vertices in V'. That is there will be at most $\frac{(a-1)a}{2}$ edges to be through-routed. Each such edge $\{u, v\} \in E'$ may pass through at most $a-2$ elements of V' so

$$\sum_{v \in V'} \sum_{e \in E'} RO_G(e, v) \leq \frac{a(a-1)(a-2)}{2} = \frac{a^3 - 3a^2 + 2a}{2}.$$

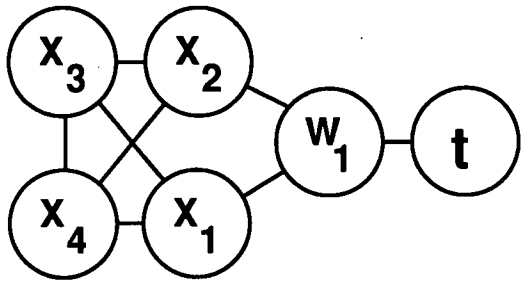


Figure 6–1: Carbuncle for Degree 3

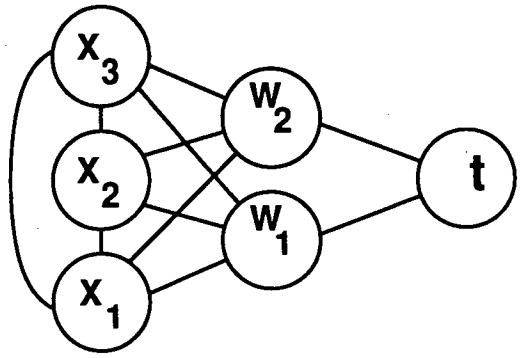


Figure 6–2: Carbuncle for Degree 4

□

We now define a function which, given a degree $\mathcal{D}$, an index k and a vertex t returns a graph structure with all vertices except t connected by $\mathcal{D}$ edges and t connected by $\mathcal{D}-2$ edges. We refer to this graph structure as a *Carbuncle.*

Given a degree $\mathcal{D}$, a vertex t and an index k, let $\chi(t,k,\mathcal{D})$ denote the pair $\langle T^k, F^k\rangle$, where $T^k = W^k \cup X^k$ defined as follows. For $\mathcal{D}=2$, $W^k = X^k = F^k = \{\}$.

For D= 3, $X^k = \{x_1^k x_2^k, x_3^k, x_4^k\}$, $W^k = \{w^k\}$, and

$$F^k = \{\{x_i^k, x_j^k\} : x_i^k, x_j^k \in X^k\} - \{\{x_1^k, x_2^k\}\} \cup \{\{x_1^k, w^k\}, \{x_2^k, w^k\}, \{w^k, t\}\}.$$

For $\mathcal{D} >= 4$, $W^k = \{w_1^k, w_2^k, \ldots, w_{\mathcal{D}-2}^k\}$, $X^k = \{x_1^k, x_2^k, \ldots, x_{\mathcal{D}-1}^k\}$ and

$$\begin{aligned} F^k \;=\; & \{\{t, w_i^k\} : w_i^k \in W^k\} \bigcup \{\{w_i^k, x_i^k\} : w_i^k \in W^k, x_i^k \in X^k\} \\ & \bigcup \{\{x_i^k, x_{i+1}^k\} : i = 1, 2, \ldots, \mathcal{D}-2\} \bigcup \{\{x_1^k, x_{\mathcal{D}-1}^k\}\}. \end{aligned}$$

Figures 6–1 and 6–2 respectively show the graphs corresponding to this definition when $\mathcal{D}=3$ and $\mathcal{D}=4$. In the case where $\mathcal{D}=2$, W^k and F^k are empty sets. In the case where $\mathcal{D}=3$, we note that the elements of $X^k \cup W^k$ are

arranged in a cycle, and w^k is adjacent to t. In the case where $\mathcal{D} \geq 4$, we note that the elements of $X^k \cup W^k \cup \{t^k\}$ are arranged in a cycle. Thus we conclude that for all $\mathcal{D} \geq 2$, the graph $\chi(t, k, \mathcal{D}) = (T^k \cup \{t\}, F^k)$ is connected.

We now show that if a graph consisting of carbuncles of degree $\mathcal{D}$ is to have degree $\mathcal{D}$ then the carbuncles must be connected in a ring or a chain.

Lemma 6.2 *Let $\mathcal{D} \geq 2$. Let T be a set of vertices. Let $t_i, \ldots, t_{|T|}$ be an enumeration of T and for each $t_i \in T$, let $\chi(t_i, i, \mathcal{D}) = \langle T^i, F^i \rangle$. Any graph $G = (V, L)$ with degree $\leq \mathcal{D}$ where $V = \bigcup_{i=1}^{|T|} T^i \cup T$ and $\bigcup_{i=1}^{|T|} F^i \subseteq L$ is connected iff the subgraph G' of G with vertex set T and edge set $L' = L - \bigcup_{i=1}^{|T|} F^i$ is a chain or a ring.*

Proof

IF We noted above that each carbuncle $(T^i \cup \{t_i\}, F^i)$, is a connected subgraph of degree $\mathcal{D}$. Thus the graph G is connected if it contains a connected subgraph with vertex set T. We note that each vertex $t_i \in T$ is adjacent to $\mathcal{D} - 2$ carbuncle edges. If these vertices are connected in a subgraph L' which is a ring or a chain, then every vertex in T is adjacent to at most 2 edges in L', and thus the degree of G is $\mathcal{D}$. Furthermore all carbuncles are connected, and we conclude G is connected.

ONLY IF Now for any carbuncle $(T^i \cup \{t_i\}, F^i)$, all vertices in T^i have degree $\mathcal{D}$ and therefore in any connected graph $G = (V, L)$ with degree $\leq \mathcal{D}$ there cannot be an edge in $L' = L - \bigcup_{i=1}^{|T|} F^i$ which is incident to a vertex in T^i. Thus for all the carbuncles to be in a single connected component, the subgraph G' of G with vertex set T and edge set $L' = L - \bigcup_{i=1}^{|T|} F^i$ must be connected. In this case, since for each $t_i \in T$ the degree of t_i is bounded by $\mathcal{D}$ and t_i is incident to $\mathcal{D} - 2$ edges in F^i, t_i can be adjacent to at most 2 other vertices in T. Thus we conclude that G' is either a ring or a chain. □

Lemma 6.3 *Let $\mathcal{D} \geq 2$. Let T be a set of vertices. Let $t_1, \ldots, t_{|T|}$ be an enumeration of T and for each $t_i \in T$, let $\chi(t_i, i, \mathcal{D}) = \langle T^i, F^i \rangle$. Let $G = (V, L)$ be a graph with*

degree $\leq \mathcal{D}$ *where* $V = \bigcup_{i=1}^{|T|} T^i \bigcup T$ *and* $\bigcup_{i=1}^{|T|} F^i \subseteq L$. *For all* $t_i, t_j \in T$,$i \neq j$, *for all* $v \in \bigcup_{i=1}^{|T|} T^i$, $RO_G(\{t_i, t_j\}, v) = 0$.

Proof

Let us assume that for some pair of vertices, say $t_i, t_j \in T$,$i \neq j$, for some $t_l \in T$, for some vertex, say $v \in T^l$, $RO_G(\{t_i, t_j\}, v) \neq 0$. Thus $v \in SH_G(\{t_i, t_j\})$. Now, by Lemma 6.2, the vertices of T are arranged in a ring, thus $t_l \in SH_G(\{t_i, v\})$ and $t_l \in SH_G(\{v, t_j\})$. This implies t_l is traversed twice in the shortest path from t_i to t_j which is in contradiction to our definition of shortest path. Thus we conclude $v \notin SH_G(\{t_i, t_j\})$ and the lemma is proved. □

The following facilities location problem is known to be NP complete [Garey *et al.* 1976].

Decision Problem 6.1 SOLA

Let $G = (V, E)$ *be a graph and* $\mathcal{K}$ *be a positive integer. Does there exist a one-to-one function,* $f : V \mapsto \{1, 2, \ldots, |V|\}$, *such that* $\sum_{\{u,v\} \in E} |f(u) - f(v)| \leq \mathcal{K}$?

We wish to show the NP completeness of the following problem.

Decision Problem 6.2 MCP

Let $J = (T, F)$ *be a graph,* $\mathcal{D} \geq 2$ *be an integer degree,* $\mathcal{B}$, *be a non-negative integer bound and* $c : F \mapsto Z_0^+$, *representing the volume of communication between a pair of tasks, say* t_i *and* t_j, *for each* $\{t_i, t_j\} \in F$. *Does there exist a graph, say* $H = (T, L)$, *such that each vertex in* T *has degree* $\mathcal{D}$ *and such that* $\sum_{e \in F} SH_H(e) \times c(e) \leq \mathcal{B}$?

Fixing $\mathcal{D} = 4$, we have the TCP: Transputer Configuration Problem, since transputers have just 4 physical links per processor [INMOS Limited, 1988].

We define the following polynomial time transformation from SOLA to TCP.

Definition 6.1 SOLA-to-TCP Encoding..

Let $G = (V, E)$ and $\mathcal{K}$ be an instance of SOLA. We construct an instance, $J = (T, F)$, $\mathcal{B}$ and c, of the problem TCP as follows.

Let $d = |V|$. Let $\mathcal{B} = \mathcal{K} - |E| + d^2 - d$. Let T_{pad} be a set of d^3 new vertices and s be a new vertex. Let $\lambda = d^3 + d$. Let $T' = V \cup T_{pad} \cup \{s\}$. Let $\{t_0, t_1, \ldots, t_d, \ldots, t_\lambda\}$ be an enumeration of T' such that $t_0 = s$; for $i = 1, 2, \ldots, d$, $t_i \in V$, and for $i = d+1, d+2, \ldots, \lambda$, $t_i \in T_{pad}$. We note that $|T'| = d^3 + d + 1$, which is always an odd number.

Let $F' = F_{pad} \cup F_\alpha \cup F_\beta \cup E$ where $F_{pad} = \{\{t_i, t_{i+1}\} : i = d, \ldots, \lambda - 1\}$, $F_\alpha = \{\{t_0, t_i\} : i = 1, \ldots, d\}$ and $F_\beta = \{\{t_0, t_i\} : i = d^3 + 1, \ldots, \lambda\}$.

For $i = 0 \ldots, \lambda$ let $\langle T^i, F^i \rangle = \chi(t_i, i, 4)$. Let $T^* = \cup_{i=0}^{\lambda} T^i$. Let $T = T^* \cup T'$, $F^* = \cup_{i=0}^{\lambda} F^i$ and $F = F^* \cup F'$.

The function, c, is defined as for each $e \in F'$, $c(e) = 1$, and for each $e \in F^*$, $c(e) = \mathcal{B} + 1$.

We now prove the following theorem.

Theorem 6.1 *The Transputer Configuration Problem is NP complete.*

Proof

First, we prove that TCP is in NP. In order to compute the sum, the non-deterministic Turing machine must find a minimum-length route between each pair of vertices , say u and v, such that $\{u, v\} \in F$. This can be done in $O(|T|^3)$ time [Aho et al., 1983]. The other operations are simple arithmetic. Thus TCP is in NP.

Let the graph $G = (V, E)$ and the positive integer, $\mathcal{K}$, be an instance of SOLA. The SOLA-to-TCP encoding method, described above, produces an instance, $J = (T, F)$, $\mathcal{B}$ and c, of TCP in an amount of time that is polynomial in the size of the instance of SOLA.

We now prove that there exists a one-to-one function $f : V \mapsto \{1, \ldots, |V|\}$, such that

$$\sum_{\{u,v\} \in E} |f(u) - f(v)| \leq \mathcal{K}$$

if and only if there exists a graph $H = (T, L)$ such that each $t \in T$ has degree 4 in H and

$$\sum_{e \in F} SH_H(e) \times c(e) \leq \mathcal{B}.$$

ONLY IF Suppose in our instance $G = (V, E)$ of SOLA there exists a one to one function such that,

$$\sum_{\{u,v\} \in E} |f(u) - f(v)| \leq \mathcal{K}.$$

Our solution to the constructed instance $J = (T, F)$, $\mathcal{K}$, c of TCP is the graph $H = (T, L)$ with L constructed as follows. We add an edge for each edge in F^* and F_{pad}, one between t_0 and t_1, one between t_λ and t_0 and, recalling that $t_1, t_2, \ldots, t_d$ is an enumeration of V, one between each pair of tasks $t_i, t_j \in V$ where $f(t_i) - f(t_j) = 1$. That is

$$L = \{\{f^{-1}(i), f^{-1}(i+1)\} : i = 1, \ldots, d-1\} \cup \{\{t_0, t_1\}, \{t_\lambda, t_0\}\} \cup F^* \cup F_{pad}.$$

We observe that the inclusion of all edges in F^* and the edges that form the vertices in T' into a ring, means that each of the vertices in H has degree 4.

For all edges $\{v, w\} \in F^*$ the edge $\{v, w\} \in L$ forms the unique shortest path between v and w in H. For all edges $\{v, w\} \in F - F^*$, $v, w \in T'$; that is the vertices of T' are in the ring. By Lemma 6.3, any shortest route between the elements of T' would include only vertices in T'. Now since $|T'|$ is odd there is a unique shortest path around the ring between any pair of vertices in T'. Thus we conclude there is a unique shortest path in H between the vertices of any edge in F.

For every edge $e \in F^* \cup F_{pad}$, we have $e \in L$ and so, $SH_H(e) = 0$. Thus

$$\sum_{e \in F^* \cup F_{pad}} SH_H(e) \times c(e) = 0.$$

The edges in F_α and F_β are through-routed in the edges of L that form a ring, thus

$$\sum_{e \in F_\alpha} SH_H(e) \times c(e) = \sum_{e \in F_\beta} SH_H(e) \times c(e) = (d^2 - d)/2.$$

For all edges $\{u,v\} \in E$, since $SH_H(\{u,v\}) \leq |f(u) - f(v)|$ and $c(\{u,v\}) = 1$, we have

$$\sum_{\{u,v\} \in E} SH_H(\{u,v\}) \times c(\{u,v\}) \leq \mathcal{K}.$$

Thus

$$\sum_{e \in F} SH_H(e) \times c(e) \leq \mathcal{K} - |E| + d^2 - d = \mathcal{B}.$$

IF Suppose there exists a graph $H = (T, L)$ such that each vertex in T has degree 4 and

$$\sum_{\{u,v\} \in F} SH_H(\{u,v\}) \times c(\{u,v\}) \leq \mathcal{B}.$$

Note that, in any solution to TCP which achieves the bound, if there existed an edge in F^* which was not in L, it would have to be routed through at least one intermediate processor, thus exceeding the bound. Thus we may assume that all edges $F^* \subseteq L$, and that each edge $\{u,v\} \in F^*$ forms the unique shortest route between u and v. Thus

$$\sum_{v \in T, e \in F^*} RO_H(e,v) = 0. \tag{6.1}$$

Note that J is connected since the vertices in $T_{pad} \cup \{t_0\}$ are connected in J in a chain, and every vertex in V is connected in J to t_0. Thus in order for H to successfully route the messages in J, H must be connected. By Lemma 6.2, in order for H to be connected, $L - F^*$ must be a ring or a chain. The chain may be connected into a ring by the addition of an edge and that addition can never extend routes. We conclude, therefore, if there exists a solution to our constructed instance of TCP which achieves the bound, there exists a solution which achieves the bound where the vertices in T' are connected in a ring.

Thus we shall assume that in H the vertices of T' are connected in a ring. Note that since there are an odd number of vertices in T' there is a unique shortest route between any two vertices in T'. Furthermore by Lemma 6.3, the

shortest route between any pair of vertices in T' contains no vertices in T^*. Thus

$$\sum_{v \in T^*, e \in F'} RO_H(e, v) = 0. \tag{6.2}$$

By combining Equations (6.2) and (6.1) we have that in any graph H which achieves the bound where the edges of T' are connected in a ring,

$$\sum_{v \in T, e \in F} RO_H(e, v) = \sum_{v \in T', e \in F'} RO_H(e, v). \tag{6.3}$$

Now let $< t_{a_1}, t_{a_2}, \ldots, t_{a_d} >$ denote the sequence of tasks in V clockwise in the ring H from t_0. Consider a graph $H' = (T', L')$, where

$$L' = \{\{t_0, t_{a_1}\}\} \cup \{\{t_{a_i}, t_{a_{i+1}}\} : i = 1, \ldots, \lambda - 1\} \cup \{\{t_{a_\lambda}, t_0\}\} \cup F^*.$$

We show that if H achieves the bound then so does H'. That is

$$\sum_{v \in T', e \in F'} RO_{H'}(e, v) \leq \sum_{v \in T', e \in F'} RO_H(e, v). \tag{6.4}$$

We note the following.

$$\sum_{v \in T', e \in F_{pad}} RO_{H'}(e, v) = \quad 0 \quad \leq \sum_{v \in T', e \in F_{pad}} RO_H(e, v). \tag{6.5}$$

$$\sum_{v \in T', e \in F_\alpha} RO_{H'}(e, v) = \quad (d^2 - d)/2 \quad \leq \sum_{v \in T', e \in F_\alpha} RO_H(e, v). \tag{6.6}$$

$$\sum_{v \in T', e \in F_\beta} RO_{H'}(e, v) = \quad (d^2 - d)/2 \quad \leq \sum_{v \in T', e \in F_\beta} RO_H(e, v). \tag{6.7}$$

We divide this proof into two cases.

CASE I: Suppose there exists $w \in T_{pad}$, such that,

$$\forall e \in E \quad RO_H(e, w) = 0. \tag{6.8}$$

By combining Equations (6.5),(6.6) and (6.7),

$$\sum_{v \in T', e \in F' - E} RO_{H'}(e, v) \leq \sum_{v \in T', e \in F' - E} RO_H(e, v).$$

Thus it remains to be proven that:

$$\sum_{v \in T', e \in E} RO_{H'}(e, v) \leq \sum_{v \in T', e \in E} RO_H(e, v). \tag{6.9}$$

Suppose, as a hypothesis to be proven contradictory, that inequality (6.9) above, does not hold. Thus, for some edge, say $\{t_{a_l}, t_{a_m}\} \in E$, there exists $t_{a_n} \in T'$, such that $RO_H(\{t_{a_l}, t_{a_m}\}, t_{a_n}) = 0$; that is, $\{t_{a_l}, t_{a_m}\}$ is not routed through t_{a_n} in H, although it is routed through t_{a_n} in H'. Without loss of generality, let $l < m$. We sub-divide this case into two sub-cases, in each sub-case obtaining a contradiction.

SUB-CASE I.1 : $l < n < m$. Since $\{t_{a_l}, t_{a_m}\}$ is not routed through t_{a_n} in H, it must be routed anticlockwise from t_{a_l}. But in that case, w lies on the shorter route between t_{a_l} and t_{a_m}, so $RO_H(\{t_{a_l}, t_{a_m}\}, w) = 1$, contradicting Equation (6.8). Thus Inequality (6.9) holds.

SUB-CASE I.2 : $n < l$ or $m < n$. It is given that in H' t_{a_n} is on the unique shortest route between t_{a_l} and t_{a_m}. That route is anticlockwise from t_{a_l} to t_{a_m}, so it has a length no less than $d^3 - 1$. But the route clockwise from t_{a_l} to t_{a_m} has a length no greater than d and would therefore be shorter, which leads to a contradiction. Thus Inequality (6.9) holds.

Thus we conclude that in either subcase Inequality (6.9) holds and so Inequality (6.4) holds in general in case I.

CASE II: Suppose that, for all $w \in T_{pad}$, $\exists e \in E$, such that $RO_H(e, w) = 1$. Thus

$$\sum_{v \in T_{pad}, e \in E} RO_H(e, v) \geq |T_{pad}| = d^3.$$

By Lemma 6.1, we have that

$$\sum_{v \in V, e \in E} RO_{H'}(e, v) \leq (d^3 - 3d^2 + 2d)/2,$$

and by combining Equations (6.5),(6.6) and (6.7),

$$\sum_{v \in T', e \in F' - E} RO_{H'}(e, v) = (d^2 - d). \tag{6.10}$$

Thus

$$\sum_{v\in T',e\in F'} RO_{H'}(e,v) \leq (d^3 - d^2 + 2d)/2 \leq d^3 \leq \sum_{v\in T',e\in F'} RO_H(e,v)$$

and Inequality (6.4) holds.

Thus we conclude that in general Inequality (6.4) holds. This means that for our constructed graph H', by Equation (6.3)

$$\sum_{v\in T,e\in F} RO'_H(e,v) \leq \mathcal{B} = \mathcal{K} - |E| + d^2 - d.$$

That is, by Equations (6.10) and (6.3), and by the definition of RO and SH,

$$\sum_{e\in E} SH_{H'}(e) = \sum_{v\in V,e\in E} RO_H(e,v) \leq \mathcal{K} - |E|.$$

Let the function f be defined as: $f(t_{a_i}) = i, \quad i = 1,\ldots,d$. Now

$$|f(t_{a_i}) - f(t_{a_j})| = |i-j| \leq SH_{H'}(a_i,a_j) + 1,$$

so

$$\sum_{u,v\in E} |f(u) - f(v)| \leq \sum_{e\in E} SH_{H'}(e) + |E| \leq \mathcal{K}.$$

Thus f is the required allocation function for the instance of SOLA. □

Note that in our reduction in any instance of the constructed TCP problem that achieves the bound, there is a unique shortest route in the processor graph between any pair of vertices that communicate in the task graph. It should be clear that our result generalises to a version of the problem where a unique path is chosen among non-unique routes, or where message traffic is split between non-unique routes. Note that Lemma 6.2 applies for any $\mathcal{D} \geq 2$, and thus in general the bounded-degree MCP problem is NP complete. Note also that in the case where $\mathcal{D} = 2$, the carbuncles are empty and so every edge in the constructed graph has a label of 1. We refer to the above problem as "simple" MCP in the spirit of [Garey et al., 1976]. We have therefore shown that simple degree-2 MCP is NP complete. The complexity of simple general-degree MCP is open.

6.4 Contention in a Scheduling Model

The above complexity result refers to a *process based* model where the cost of a message transfer is assumed to be independent of the exact timing of communication events. A similar approach was taken by El-Rewini and Lewis [1990] for *scheduling models*, but in these models we are able to capture the timing of communication events and the way in which they interact.

One example of this approach is the work of Mak and Lundstrom [1990]. They consider the timing of communications events that results from a particular schedule and the corresponding demands that are made upon an interprocessor communications network. These demands are fed into a queuing network model which is then solved to give the expected latencies of the various communications which are fed back into the scheduling model to alter the timings of communications events. The algorithm is applied iteratively until a predefined convergence property is satisfied.

In this section we present complexity results for an explicit model of communication contention which is not based upon queuing theory. We consider that the interprocessor communications medium is a "processor" and that precedence arcs give rise to communications "tasks" that must be scheduled on the interprocessor communications medium. We shall, however, make an important assumption, that the interprocessor communications cannot be buffered. That is, if a task executed on one processor has an outgoing precedence arc to a task executed on another processor, the first processor must remain idle following the first task's execution until a task corresponding to the interprocessor communication is executed by the communications medium.

Intuitively our model corresponds to a communication system with no buf-

fering at the sender, such as Meiko's CS-Tools [Meiko 1992] in synchronous transfer mode with blocking send and non-blocking receive.

6.4.1 Relevant Complexity Results

Our complexity result is for UET tasks in a two processor system with a single bidirectional communications link between the processors. As mentioned in Section 4.1, the complexity of two processor scheduling is open in UET models with *General Delay, Uniform Delay, Fixed Delay* and *Unit Delay*. There is a polynomial time algorithm for the problem in UET *No Delay* models [Coffman and Graham 1972].

Since we are effectively introducing an extra processor, corresponding to the interprocessor communications medium, three processor scheduling complexity results are also of interest. The complexity of scheduling is open in all delay-variants upon 3 processor UET scheduling models. Our final related result is that by Berger and Cowen [1991] who show the NP completeness of scheduling in 3 processor models where UET tasks may have to be executed simultaneously on two processors.

6.4.2 The Complexity of Scheduling with Contention

Definition 6.2 Two-Processor UET Contention Model and Schedule.
Given an instance of a unit delay UET scheduling model, $(P, (\Gamma, \Delta), f, d)$, where $|P| = 2$, an interprocessor communications link π and a communication event κ, we can construct a corresponding instance, say B, of a 2 processor UET contention model. Let $B = (P \cup \{\pi\}, (\Gamma \cup \{\kappa\}, \Delta), \pi, \kappa)$. For all $\gamma \in \Gamma \cup \{\kappa\}$, let $f'(\gamma) = 1$. A schedule for B is a schedule of $(P \cup \{\pi\}, (\Gamma \cup \{\kappa\}, \Delta), f', d)$.

The definition of what constitutes a valid schedule in such a model is slightly different from that in scheduling models outlined earlier. We refer to it as *con-*

tentionally valid to avoid confusion, and define it more formally below. Note that one of the conditions of contentional validity is that the subset of the schedule involving only processors (rather than interconnect) is valid with respect to a corresponding *unit delay* scheduling model. The other conditions of validity are that the schedule as a whole is not overbooked, that computational tasks are not scheduled for execution on the interconnect, and that for every precedence arc between tasks mapped to different processors there is a corresponding interprocessor communication event in the schedule.

Definition 6.3 Contentionally Valid Schedule.
Let $B = (P', (\Gamma', \Delta), \pi, \kappa)$, be a 2 processor UET contention model. Let f be a function with domain $\Gamma' - \{\kappa\} \times P' - \{\pi\}$ and range $\{1\}$. Let d be a function with domain $P' - \{\pi\}$ and range $\{1\}$. Let $A = (P' - \{\pi\}, (\Gamma' - \{\kappa\}, \Delta), f, d)$. Let s be a schedule for B. Let $s' = X(s, \Gamma, P, Z_0^+)$. Let $s^* = X(s, \Gamma, \{\pi\}, Z_0^+)$

s is *contentionally* valid if all of the following are true.

- s' is valid for A.
- s is not overbooked.
- $s^* = \{\}$.
- there exists a valid tuple dependence graph (s', C) for s' such that for all *interprocessor* dependences $((\gamma, p, t)(\delta, q, t')) \in C$ there exists a tuple $(\kappa, \pi, t+1) \in s$.

We show the NP completeness of the following problem by reduction from problem (P5) in [Ullman 1975] which is stated above as Decision Problem 3.4. Our complexity proof, like that of Theorem 3.7 above, follows the outline of Ullman's proof of the complexity of his problem (P3).

Decision Problem 6.3 *Let $B = (P', \Lambda' = (\Gamma', \Delta'), \pi, \kappa)$ be a 2 processor UET contentional model. Let n' be a positive integer. Does there exist a valid contentional schedule s' for B such that $\mathcal{M}(s') = n'$?*

Theorem 6.2 *Decision Problem 6.3 is NP complete.*

Proof

Suppose we are given an instance of Decision Problem 3.4 as above. We construct an instance of Decision Problem 6.3 in the way outlined in the formulae below. To clarify the notation, the tasks of Ξ and the edges of Δ'_1 form one chain, while the tasks of Υ and the edges of Δ'_2 form another. Δ'_3 is a set of edges which connect the chains as indicated in Figure 6–3. In addition we have the tasks of Γ, and for each task in Γ an edge in Δ'_4 to a corresponding task in Γ^*. In Δ'_5, for every edge $(\alpha, \beta) \in \Delta$ we have an edge from α to the vertex corresponding to β in Γ^*. Finally, every vertex in Γ^* has an edge to the last vertex in the chain formed by the elements of Ξ.

- $P' = \{p_1, p_2, \pi\}$
- $n' = 8nk$
- $\Lambda' = (\Gamma', \Delta')$ where
- $\Gamma' = \Xi \cup \Upsilon \cup \Gamma \cup \Gamma^* \cup \{\kappa\}$ where
  - $\Xi = \{\Xi_{h,i,j} : h = 0, \ldots, 3, i = 0, \ldots, 2k-1, j = 0, \ldots, n-1\}$.
  - $\Upsilon = \{\Upsilon_{h,i,j} : h = 0, \ldots, 2, i = 0, \ldots, 2k-1, j = 0, \ldots, n-1\}$.
  - $\Gamma^* = \{\gamma^*_i : i = 0, \ldots, nk-1\}$.
- $\Delta' = \Delta'_1 \cup \Delta'_2 \cup \Delta'_3 \cup \Delta'_4 \cup \Delta'_5 \cup \Delta'_6$ where
  - $\Delta'_1 = \{(\Xi_{h,i,j}, \alpha) : \Xi_{h,i,j} \in \Xi - \{\Xi_{3,2k-1,n-1}\}$, if $h < 3$ then $\alpha = \Xi_{h+1,i,j}$ else if $i < 2k-1$ then $\alpha = \Xi_{0,i+1,j}$ else $\alpha = \Xi_{0,0,j+1}\}$

- $\Delta'_2 = \{(\Upsilon_{h,i,j}, \alpha) : \Upsilon_{h,i,j} \in \Upsilon - \{\Upsilon_{2,2k-1,n-1}\}$, if $h < 2$ then $\alpha = \Upsilon_{h+1,i,j}$ else if $i < 2k-1$ then $\alpha = \Upsilon_{0,i+1,j}$ else $\alpha = \Upsilon_{0,0,j+1}\}$
- $\Delta'_3 = \{(\Xi_{0,i,j}, \Upsilon_{1,i,j}) : i = 0, \ldots, 2k-1, j = 0, \ldots, n-1\} \cup$
  $\{(\Xi_{1,i,j}, \Upsilon_{2,i,j}) : i = 0, \ldots, k-1, j = 0, \ldots, n-1\} \cup$
  $\{(\Upsilon_{1,i,j}, \alpha), (\Upsilon_{2,i,j}, \beta) : i = 0, \ldots, 2k-1, j = 0, \ldots, n-2,$
  if $i < 2k-1$ then $\alpha = \Xi_{0,i+1,j}, \beta = \Xi_{1,i+1,j}$
  else if $j < n-1$ then $\alpha = \Xi_{0,0,j+1}, \beta = \Xi_{1,0,j+1}$,
  else we have no edge$\}$
- $\Delta'_4 = \{(\gamma_i, \gamma_i^*) : i = 0, \ldots, nk-1\}$.
- $\Delta'_5 = \{(\gamma_i^*, \gamma_j) : (\gamma_i, \gamma_j) \in \Delta\}$.
- $\Delta'_6 = \{(\gamma_i^*, \Xi_{3,2k-1,n-1}) : i = 0, \ldots, nk-1\}$.

First observe that $|\Gamma'| = 16kn$, and since our deadline is $8kn$ and we have two processors, any valid schedule is a *perfect schedule* with a single tuple instancing each task in Γ'.

Next note that because of the edges Δ'_1 one processor must be devoted to processing an element of Ξ at each time unit if the time limit is to be met, thus the tasks of Ξ in the order indicated by the precedence relation Δ'_1 form the *critical path* of the computation. Moreover, the same processor must process all elements of Ξ since no communication latency can be allowed to occur between the execution of the elements of the critical path if the computation is to finish by the deadline. Without loss of generality let us assume that it is processor p_1 that is computing the tasks in Ξ, and therefore in any valid schedule s of our instance of Decision Problem 6.3, for all $\Xi_{h,i,j} \in \Xi$, $X(s, \{\Xi_{h,i,j}\}, P, Z_0^+) = \{(\Xi_{h,i,j}, p_1, t_{h,i,j})\}$, where $t_{h,i,j} = h + 4i + 8kj$. and for all $\gamma \in \Gamma' - \Xi$, $X(s, \{\gamma\}, P, Z_0^+) = \{(\gamma, p_2, t_\gamma)\}$.

The tasks in Υ must be executed at very specific times as shown in Figure 6–3. Note that tasks $\Upsilon_{0,i,j}, i = 0, \ldots, 2k-1, j = 0, \ldots, n-1$ may, if we only consider the precedence relation between tasks in $\Xi \cup \Upsilon$, be executed in either

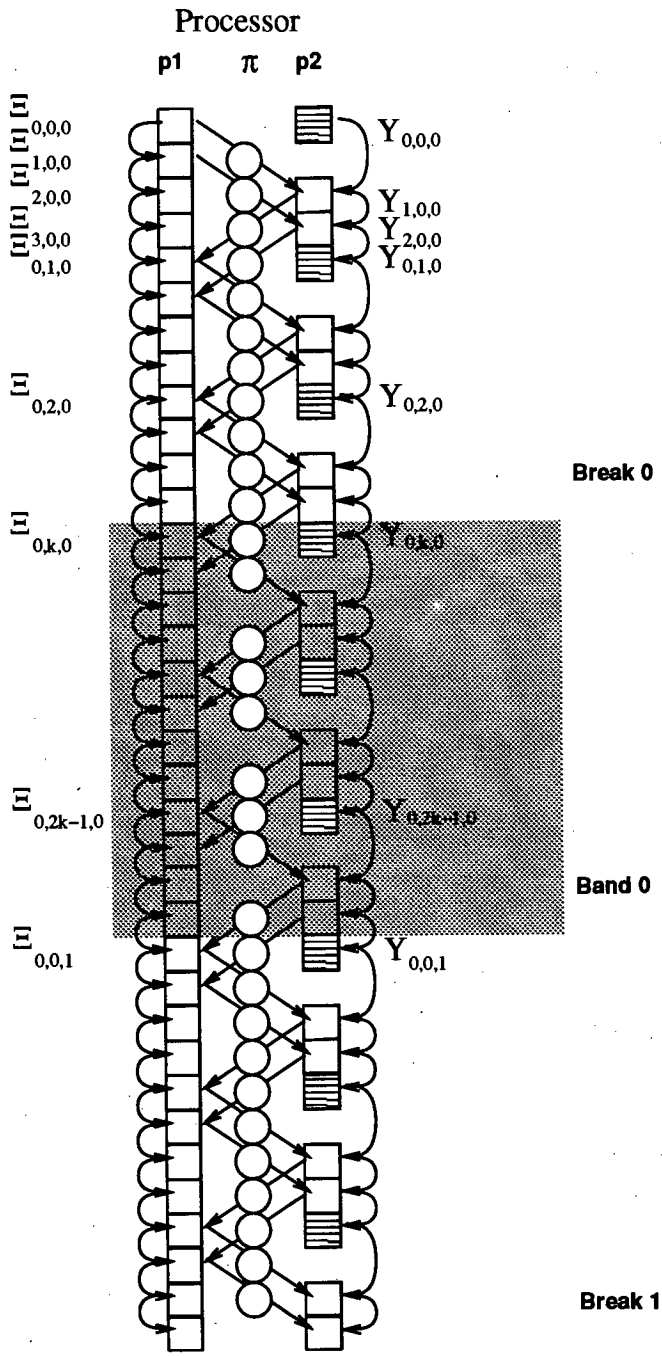


Figure 6–3: Partial Gantt Chart for the Encoding in Decision Problem 6.3

the time step shown or the subsequent time step. Note that, progressing in time, we have on processor p_2, an alternation of *breaks* in which there are k available time slots where it is *not* possible for a task to communicate in the subsequent step, and *bands* in which there are k available time slots where it *is* possible for a task to communicate in the subsequent step. Note that the kn tasks in Γ^* must communicate to task $\Xi_{3,2k-1,n-1}$ which is scheduled on processor p_1. Now since we must never have an idle time slot on any processor, the tasks of Γ^* must be executed in the kn available slots in *bands*, and tasks $\Upsilon_{0,i,j}, i = k, \ldots, 2k-1, j = 0, \ldots, n-1$, *must* be executed at the times shown. This also means the kn tasks in Γ must be executed in the kn available slots in *breaks*.

As a consequence, if tasks $\gamma, \delta \in \Gamma$ are both executed in the same break it is not possible for there to exist an edge (γ, δ) in Δ. For if so, by our construction there would exist a task $\alpha \in \Gamma^*$ such that there exist edges (γ, α) and (α, δ) in Δ' and α would have to be executed in that same break, violating what we have just concluded. Thus if our instance of Decision Problem 6.3 has a solution, we can find a solution to the original instance of Decision Problem 3.4 by executing at time unit i exactly those jobs executed in the ith break.

Conversely if we have a solution to the given instance of Decision Problem 3.4 we can find a solution of the constructed instance of Decision Problem 6.3 by executing γ_i^* in band t and γ_i in break t whenever γ_i is executed at time unit t in the solution to Decision Problem 3.4. □

Note that the above proof establishes that the problem remains NP complete even if recomputation is allowed, and even if a single interconnect event is sufficient for any number outgoing edges from a task. Note that the reduction no longer works in a case corresponding to non-blocking communication where it is assumed that outgoing communications can be indefinitely buffered.

Chapter 7

Hybrid Models

In this this chapter we consider models in which communication costs and delays are integrated, at least to some extent. As well as surveying the field, one of the aims of this chapter is to present models that will be used later to analyse a particular mapping problem. We describe models by Papadimitriou and Ullman in which considerations of communication costs are included into a *No Delay* scheduling model and we propose a new model which integrates communications costs into a *General Delay* scheduling model. We also consider general purpose scheduling algorithms for this new model.

7.1 Integrating Computation and Communication

In Bokhari's model, the computation cost of each process is assumed to be equal and one process is mapped to each processor. This leads to an optimal load balance, but it may well be that the best mapping, once communications costs are taken into consideration, is not amongst those which optimally balance load. Considerable research effort has gone into extending the two process based models discussed in Chapter 5, namely Stone's approach of assuming sequential execution and Bokhari's approach of considering only communication overheads.

The basic problem of adding communication costs to computation costs is twofold. First, as Efe points out, they may be measured in different units [Efe 1982][1]. Second, the cost of a communication is attributable to a particular processor or pair of processors and not to the calculation as a whole.

---

[1]Recall that in the case of scheduling models with delays we were able to solve this problem because message lengths could be turned into times by considerations of communications *bandwidth*.

7.1.1 Mini-Max formulations

There is a class of models which may be thought of as versions of Stone's model where costs are not incurred sequentially, or of Bokhari's models where more than one process may be assigned to the same processor. Indurkhya *et al.* [1986], who consider randomly generated programs, simply add all communications costs to the maximum of the processors' execution costs – defined as the sum of the costs of the computations assigned to each processor. See Nicol [1989] for the limitations of the results in this paper. A variation on this is the work of Efe [1982] who does not explicitly propose this model but whose mapping algorithm iteratively improves this quantity from an initial heuristic assignment of processes.

Bokhari [1988] describes polynomial time algorithms for solving the mini-max problem in the case of chains of processes and chains of processors with the constraint that communicating processes must be mapped to adjacent processors, and also in various other constrained process formulations for host-satellite processor systems.

Shen and Tsai [1985] use a graph matching algorithm to minimise costs in a model which assigns communication costs to both the sending and receiving processor. They state that they are considering computations where "little or no" precedence relation exists between processes. As a result they assume negligible processor idleness (T_{Wait} in our framework), and their communication costs can be associated directly with T_{Init} and T_{Term} which are added respectively to the computation overhead of the sending and receiving processors. This model may be appealing to the multicomputer programmer, but it is important to remember that on a multicomputer T_{Init} and T_{Term} are rarely dependent upon the distance in the processor network over which a particular message is to be sent, and so it may be that the processor dependence of communication cost is being inadequately modelled by Shen and Tsai.

Chu and Lan [1987] use a very similar model to Shen and Tsai, but work with a probabilistic task based model rather than the corresponding process based model. They consider the effect of the probabilistic precedence relation upon the time to completion and conclude that, where two processes are connected by a precedence relation, if the cost of execution of the second process is much larger than that of the first process, then they should be mapped to the same processor, whereas if the cost of execution of the second is much smaller than the first, they should be mapped to different processors. They use this heuristic, and one which tends to group heavily communicating processes, to form grains which are mapped to the same processor, and then map the grains to processors by an exhaustive search which minimises costs ignoring precedence. Both Shen and Tsai [1985] and Chu and Lan [1987] allow the possibility of re-computation of processes on processors so as to minimise the overall execution time.

7.1.2 Papadimitriou and Ullman's 1987 Models

Papadimitriou and Ullman [1987] propose two models of parallel computation. Each model contains two components: the processing time of the computation, and the communication requirement of the computation. Papadimitriou and Ullman are careful never to add the time and communication quantities together, merely claiming that in situations where communications are expensive one would use a schedule where the communications cost is minimal (such as one where all tasks are assigned to the same processor), and that in situations where it is cheap it would be more appropriate to use a schedule with many communications in order to minimise the computation time of the schedule.

The two models differ in their notion of communication cost. In the *event* model the communication corresponding to every directed arc from a task allocated to one processor to a task allocated to another processor incurs a cost, analogous to paying the price of a postage stamp.

In the model they refer to as *delay*, but which we refer to as the *chain* model to avoid confusion with the delay based scheduling models, it is the number of *dependent* communications that causes the problem.

As discussed in Chapter 8 below, Papadimitriou and Ullman consider properties of something referred to as partitions rather than of schedules. For our purposes, we can define Papadimitriou and Ullman's costs with reference to a tuple dependence graph of Definition 2.34.

Definition 7.1 Event Cost, Chain Cost.
Let $D = (s, C)$ be a tuple dependence graph. Let C' be the subset of C containing all interprocessor tuple dependences. The *event cost* of D, denoted $\mathcal{E}(D)$, is given by

$$\mathcal{E}(D) = |C'|.$$

The *chain cost* of D, denoted $\mathcal{C}(D)$, is given by

$$\mathcal{C}(D) = \max_{X \text{ is a path in } D} |X \cap C'|.$$

7.2 Claud

In this section we propose a new hybrid model whereby communication costs and delays are treated in a consistent way. The basis of the model is that communication is subject to a delay, and also incurs a cost in the form of two computational *tasks* referred to as *communications events*. These must be scheduled, one by each of the sending and the receiving processor. The tasks involve respectively setting up the transfer of a message and setting up the receipt of a message.

Claud schedules allow processors to be in one of four states: computing, sending a message, receiving a message and idle. The schedule corresponds to

a process time graph (Lamport [1979]), as annotated by Lo [1992] where each processor is executing a single process. Related approaches have been described in the partitioning literature. Agrawal and Jagadish [1988] use a model of scheduled overheads of both send and receive type, but deal only with independent tasks (ie. those without precedence relations). McCreary and Gill consider minimising communications costs by partitioning communications into messages, but they do not consider these costs as schedulable computations–rather they simply add computation costs and communications costs to get the overall cost of the partition. Finally, Kruatrachue and Lewis [1988] describe an approach that they refer to as Grain Packing, whereby communications costs are aggregated within a grain, but give insufficient details to determine the relationship with our approach.

A Claud schedule consists of tuples corresponding to the execution of tasks in the original dag and also to the special communications tasks. The conditions of validity of the schedule are similar to those outlined in Section 2.2 for scheduling models, but with the added complication that communications must be packaged into messages. The three conditions are

1. All tasks in the original dag are executed.

2. No processor is executing more than one task at a time, where that task may be either in the dag or a communications task.

3. The tasks that are defined to precede a given task have finished executing before that task is started, and if they are executed on different processors, a message has been passed between the two processors, into which it was possible to fit the data corresponding to the precedence arc.

In the following we formally define a message as a set of tuple dependences where all the start tuples have been mapped to the same processor and all the finish tuples have been mapped to the same processor.

Definition 7.2 (Valid) Message.

Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a general delay scheduling model, and s be a valid schedule for A. Let $D = (s, C)$ be a valid tuple dependence graph for s. A *message,* say m, is a non-empty subset of C such that for each pair of tuples $((\gamma, p, t), (\delta, q, r)), ((\gamma', p', t'), (\delta', q', r')) \in m$, $p = p'$ and $q = q'$. We define the *length,* denoted $\mathcal{L}(m)$, the *release time,* denoted $\mathcal{R}(m)$, and the *deadline,* denoted $\mathcal{D}(m)$ of m in the following way.

$$\begin{aligned} \mathcal{R}(m) &= \max_{((\gamma,p,t),(\delta,q,r))\in m} t + f(\gamma) \\ \mathcal{D}(m) &= \min_{((\gamma,p,t),(\delta,q,r))\in m} r \\ \mathcal{L}(m) &= \sum_{((\gamma,p,t),(\delta,q,r))\in m} d((\gamma,\delta),p,q) \end{aligned}$$

m is a *valid message* for D iff $\mathcal{R}(m) + \mathcal{L}(m) \leq \mathcal{D}(m)$.

Definition 7.3 Valid Message Partition.

Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a general delay scheduling model, and s be a valid schedule for A. Let $D = (s, C)$ be a tuple dependence graph for A. Let C' be the subset of C containing all interprocessor tuple dependences. Let $m = \{m_1, \ldots, m_x\}$ be a partition of C. m is a *valid message partition* for A iff each $m_1, \ldots, m_x$ is a valid message for D.

Definition 7.4 Claud Model.

Given an instance $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ of a *General Delay* scheduling model and two tasks, σ and ρ which are involved in respectively sending and receiving a message and which respectively require time λ_σ and λ_ρ for execution, we construct an instance B of a general delay *Claud* scheduling model as follows. For all $\gamma \in \Gamma$, let $f'(\gamma) = f(\gamma)$. Let $f'(\sigma) = \lambda_\sigma$. Let $f'(\rho) = \lambda_\rho$. Let $B = (P, \Lambda, f', d, \lambda_s, \lambda_r)$. B is a *Unit Delay* Claud model if A is a *Unit Delay* scheduling model. B is a *No Cost* Claud model if $\lambda_\sigma = \lambda_\rho = 0$.

Definition 7.5 Claud Schedule.

Let $B = (P, \Lambda = (\Gamma, \Delta), f', d, \lambda_s.\lambda_r)$ be a Claud model. A *Claud schedule* of B is a schedule of $(P, (\Gamma \cup \{\lambda_s, \lambda_r\}, \Delta), f', d)$.

Definition 7.6 Support for a Message Partition.
Let $A = (P, \Lambda = (\Gamma, \Delta), f', d, \sigma, \rho)$ be an instance of a Claud scheduling model. Let s be a Claud schedule for A. Let f be the restriction of f' to the domain Γ. Let $s' = X(s, \Gamma, f, d)$. Let D be a tuple dependence graph for s and $m = \{m_1, m_2, \ldots, m_x\}$ be a valid message partition for D. let $S = X(s, \{\sigma\}, P, Z_0^+)$. Let $R = X(s, \{\rho\}, P, Z_0^+)$.

s is said to *support* m if there exist two one to one functions $\mathcal{F}_s : m \rightarrow S$ and $\mathcal{F}_r : m \rightarrow R$ such that for each $m_i \in m$, where say $(\gamma, p, t) = f(m_i)$, $\mathcal{R}(m_i) \leq t$ and $\mathcal{D}(m_i) \geq t$.

Definition 7.7 Valid Claud Schedule.
Let $A = (P, \Lambda = (\Gamma, \Delta), f', d, \sigma, \rho)$ be an instance of a Claud scheduling model. Let f be the restriction of f' to the domain Γ. Let $B = (P, \Lambda, f, d)$. Let s be a Claud schedule for A. Let $s' = X(s, \Gamma, P, Z_0^+)$. s is *valid* iff all of the following hold.

- s it is not overbooked.
- s' is valid for B.
- There exists a valid tuple dependence graph (s', C) for s' for which there exists a valid message partition M such that that s supports M.

7.2.1 Claud and the Mapping Problem Framework

Let $A = (P, \Lambda = (\Gamma, \Delta), f', d, \sigma, \rho)$ be an instance of a Claud scheduling model. Let s be a Claud schedule for A. We can present s in terms of the framework outlined in Section 1.3 in the way shown in table 7–1.

Here all the non-communication tasks mapped to processor p contribute to $T_{Calc}(p)$ or $T_{Recalc}(p)$. Communication send tasks contribute to $T_{Init}(p)$ and communication receive tasks contribute to $T_{Term}(p)$.

T_{Calc}		$\sum_{\gamma \in \Gamma} f(\gamma)$
T_{Comm}	$T_{Init}(p)$	$f'(\sigma)\|X(s,\{\sigma\},\{p\},Z_0^+)\|$
	$T_{Term}(p)$	$f'(\rho)\|X(s,\{\rho\},\{p\},Z_0^+)\|$
	T_{Route}	0
T_{House}	T_{Recalc}	$\sum_{(\gamma,p,t)\in X(s,\Gamma,P,Z_0^+)} f(\gamma) - T_{Calc}$
	T_{Inter}	0
	T_{Sched}	0
T_{Idle}	$T_{Wait}(p)$	$\mathcal{M}(X(s,\Gamma,\{p\},Z_0^+)) - \sum_{(\gamma,p,t)\in X(s,\{\gamma\},\{p\},Z_0^+)} f(\gamma)$
	$T_{Fin}(p)$	$\mathcal{M}(s) - \mathcal{M}(X(s,\Gamma,\{p\},Z_0^+))$

Table 7–1: A Claud Model in Terms of our Framework

7.2.2 Complexity Results for Claud

The general case decision problem for scheduling in a Claud model is as follows:

Decision Problem 7.1 *Given an instance $A = (P, \Lambda, f, d, \sigma, \rho)$ of a Claud model and some integer k does there exist a valid Claud schedule s of A such that $\mathcal{M}(s) < k$?*

In the case where $f(\sigma) = f(\rho) = 0$, Decision Problem 7.1 reverts to the decision problems outlined in Chapter 3 since zero-length send and receive events may precede and succeed any tuple dependence arcs.

7.2.3 Performance Guarantees for Claud Scheduling Algorithms

In this section we consider the way in which performance guarantees may be attained by scheduling algorithms for Claud models. We consider algorithms for UET *No Delay Unit Cost* Claud models which have l processors and l tasks, with the performance guarantees given by ETF in a *No Delay* scheduling model.

Let $A = (P, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a *No Delay* UET scheduling model constructed as follows.

$$\begin{aligned} P &= \{p_i : i = 1, \ldots, l\}, \\ \Gamma &= \{\gamma_i : i = 1, \ldots, l\}, \\ \Delta &= \{(\gamma_i, \gamma_l) : i = 1, \ldots, l-1\}. \end{aligned}$$

Let s^A_{ETF} be the schedule produced by ETF on A whereby nondeterminism is resolved by allocating the task of lowest index to the processor of lowest index. That is,

$$s = \{(\gamma_i, p_i, 0) : i = 1, \ldots, l-1\} \cup \{(\gamma_l, p_1, 2)\}.$$

Note that $\mathcal{M}(s^A_{ETF}) = 2$.

Note that, in s^A_{ETF}, data satisfying all but one of the $l-1$ incident edges to γ_l are being simultaneously received by processor p_i. In Claud (and we would venture to suggest in all existing multicomputers) each of those communication events must incur an overhead which is modelled as a task which is scheduled on processor p_j.

Let $B = (P, \Lambda = (\Gamma, \Delta), f', d, \sigma, \rho)$ be an instance of a UET *No Delay Unit Cost* Claud scheduling model with Λ as given above. Let s be a valid Claud schedule for B whereby k processors are involved in computing the tasks in $\gamma_1, \ldots, \gamma_{l-1}$. At least one of the k processors must compute at least $(l-2)/k$ tasks and send a message and the processor computing γ_i must receive at least $k-1$ messages. Thus

$$\mathcal{M}(s) \geq (l-2)/k + \lambda_\sigma + \lambda_\rho(k-1) + 1$$

and thus $\mathcal{M}(s) \geq (l-2)/k + 1 + k$.

Differentiating with respect to k we find that $(l-2)/k + 1 + k$ has a minimum at $l - 1 = \sqrt{k}$. Thus

$$\mathcal{M}(s) \geq (l-2)/\sqrt{l-1} + 2 + \sqrt{l-1}.$$

That is, although the ETF algorithm for mapping in *scheduling* models has a performance guarantee that is independent of the *width* of the dag, it is not possible to achieve an equivalent bound in a *Claud* model. Returning to the theme of the thesis, we wish to stress that Claud models' relatively poor algorithmic guarantees result from their modelling extra features of the multicomputer, rather than from defects in the models. As we will see in Chapter 9 this added *verisimilitude* leads to increased predictive power.

Chapter 8

Applying the models

This chapter and Chapter 9 consider the way in which the models of computation outlined in previous chapters can be used to predict the performance of a computation. In this chapter we show that the different models produce qualitatively different predictions of performance. In Chapter 9 we show the degree to which the models predict the performance of a real computation running on a real multicomputer.

8.1 The Diamond dag

As we saw in, for example Chapters 2 and 7, some parallel computations can be represented as a dag in which the nodes represent sequential subcomputations and there is an arc from a node v to a node w if the output from the operation performed at v is needed as one of the inputs to the operation at w. In this chapter and the subsequent one we consider computations that can be expressed as *indexed families* of dags, whereby the indices refer to some aspect of "problem size".

Examples of indexed families of dag computations are the fast Fourier transform graph, the *binary tree* for arithmetic expression evaluation and the *diamond* for computing the longest common subsequence [Aho *et al.* 1976] or for the numerical computation which is described in Section 9.1.

The $n \times n$ diamond is most simply described if we imagine vertices occupying points in cartesian space from $(0,0)$ in the lower left hand corner. We shall refer to the vertices as $\Gamma = \{\gamma_j^i : i = 0, \ldots, n-1, j = 0, \ldots, n-1\}$. Where n is the index of the diamond as referred to above. The 6×6 diamond is shown in Fig. 8–1[1]. The arcs in the diamond are as follows:

[1]We choose to break with the convention that in diagrams of the diamond, the start

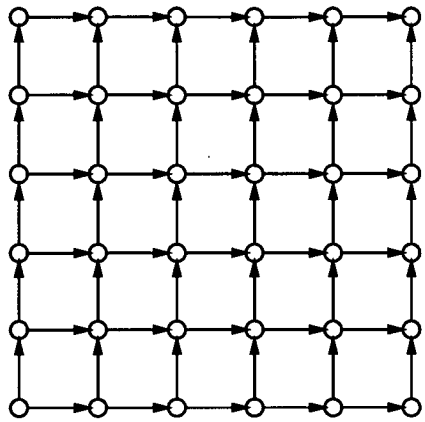


Figure 8–1: The 6×6 Diamond dag

- the vertex γ_0^0 is the start vertex and has no incoming edges.
- vertices $\gamma_0^i, 1 \leq i < n$ each have a single predecessor, namely the vertex γ_0^{i-1}.
- vertices $\gamma_j^0, 1 \leq j < n$ each have a single predecessor, namely γ_{j-1}^0.
- all other vertices, $\gamma_j^i, 1 \leq i < n, 1 \leq j < n$, have two predecessors, γ_j^{i-1} and γ_{j-1}^i.

We examine five approaches to analysing the performance of the diamond dag computation. All five models are variants of models that have been described in previous chapters.

The first two models are *No Delay* scheduling models according to Definition 2.20, but with communication costs included, and were described by Papadimitrou and Ullman [1987], and outlined in Section 7.1.2. The third is a

---

node is at the bottom and the end is at the top. That is we rotate the cartesian grid 45 degrees clockwise.

Fixed Delay scheduling model as, for example was described by Papadimitriou and Yannakakis [1992]. The aim of the analysis is to show that the three models produce quantitatively different predictions, rather than to allow comparison with a real multicomputer.

The fourth model used to analyse performance of the diamond dag computation is a *Fixed Delay No Cost* Claud model partitioning, and the final approach is a *Fixed Delay* Claud model.

8.2 Partitions of the Diamond dag

Although, in some sections of their analysis, Papadimitriou and Ullman use a tuple-based notation similar to ours, most of their work is framed in terms of *partitions* of the diamond dag, whereby each task is mapped to a single processor. They consider two partitions: stripes and boxes. In addition, we consider a third: lines.

8.2.1 Stripes

The stripes approach is shown in Figure 8–2. Informally, if there are k processors, the diamond is divided into k stripes of width n/k. The first processor computes nodes γ_0^0 to $\gamma_{n/k-1}^0$. After n/k time units, it reaches the stripe boundary and, after a delay of τ time steps, the second processor can begin. Meanwhile, the first processor works on the second row of its stripe, and so on up the stripe. For the purposes of Section 8.4 below, we define the stripes schedule in terms of a slightly more general type of schedule which we refer to as the *holey stripes* schedule.

The *holey stripes* schedule differs from the stripes approach in that each processor, on a number of occasions during the execution of its stripe, halts

for a while before continuing with the rest of the rows. In a later section we will introduce schedules where it is executing something else during these idle periods. These periods are referred to as holes in the schedule. A given holey stripes schedule will have a *hole count*, referred to as m below, and a *hole width*, referred to as l below.

The holey stripes schedule of an $n \times n$ diamond dag is the same as the stripes schedule outlined above, except each processor halts for l time steps after processing each n/m rows of its stripe. In addition, for all processors except the first, the time at which the processor starts its stripe is delayed. Instead of starting τ time steps after the first row of the previous stripe is executed, a processor must wait $\tau n/m$ time units after the first hole in the schedule of the processor computing the previous stripe. Since we allow $l = 0$ and $n = m$, a stripes schedule is simply a special case of a holey stripes schedule.

Definition 8.1 Holey Stripes Schedule.
Let l be a non-negative integer hole width and m be a positive integer hole count. Let $A = (P = \{p_0, \ldots, p_{k-1}\}, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a *Fixed Delay* UET scheduling model, where Λ is an $n \times n$ diamond dag, and the range of d is $\{0, \tau\}$. Let $Y = (l + n/m\tau + n^2/mk)$. Let $Z = (l + n^2/mk)$. For each $i = 0, \ldots, n-1$, Let $p^i = p_{\lfloor ik/n \rfloor}$. For each $i = 0, \ldots, n-1$,for each $j = 0, \ldots, n-1$ let

$$t_i^j = i \bmod n/k + (j \bmod n/m)n/k + \lfloor ik/n \rfloor Y + \lfloor jm/n \rfloor Z.$$

We define s, the *holey stripes schedule* of A with hole width l and hole count m in the following way.

$$s = \{(\gamma_i^j, p^i, t_i^j) : i = 0, \ldots, n-1, j = 0, \ldots, n-1\}.$$

Definition 8.2 Stripes Schedule.
Let $A = (P = \{p_0, \ldots, p_{k-1}\}, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a *Fixed Delay* UET

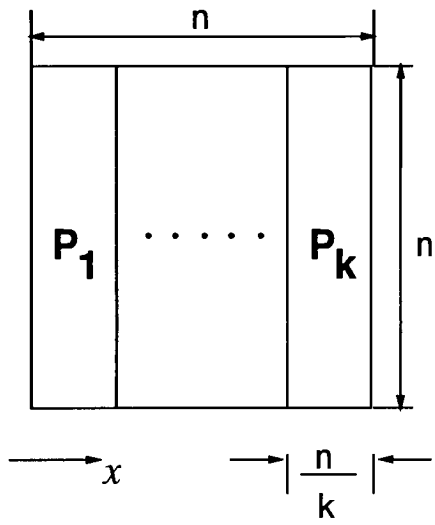


Figure 8–2: Stripes Schedule in k Processors

scheduling model, where Λ is an $n \times n$ diamond dag, the range of d is $\{0, \tau\}$ and $n/k \in Z_0^+$. Let s be the holey stripes schedule of A with hole count n and hole width 0. s is a stripes schedule for A.

Theorem 8.1 *Let m be a positive integer. Let l be a non-negative integer. Let $A = (P = \{p_0, \ldots, p_{k-1}\}, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a Fixed Delay scheduling model, where Λ is an $n \times n$ diamond dag, the range of d is $\{0, \tau\}$, $n/k \in Z_0^+$ and $n/m \in Z_0^+$. Let s be a holey stripes schedule of A with hole count m and hole width l. s is valid.*

Proof

s is complete since, by construction, there is a tuple $(\gamma_i^j, p_i, t_i) \in s$ for each $i = 0, \ldots, n-1, j = 0, \ldots, n-1$. Let us assume as a hypothesis to be proved contradictory that s is overbooked. Thus there exist two tuples, say $(\gamma_i^j, p, t) \in s$ and $(\gamma_u^v, p', t') \in s$ such that $p = p'$ and $t = t'$ and either one or both of $i \neq u$, $i \neq j$.

Let $Y = (l + n/m\tau + n^2/mk)$. Let $Z = (l + n^2/mk)$. Note by the construction of s, $p = p^{\lfloor ik/n \rfloor}$ and $p' = p^{\lfloor uk/n \rfloor}$, thus $\lfloor ik/n \rfloor = \lfloor uk/n \rfloor$ and thus $|i - u| = |i \bmod n/k - u \bmod n/k| < n/k$. Note, also that

$$\begin{aligned} t &= i \bmod n/k + (j \bmod n/m)n/k + \lfloor ik/n \rfloor Y + \lfloor jm/n \rfloor Z. \\ t' &= u \bmod n/k + (v \bmod n/m)n/k + \lfloor uk/n \rfloor Y + \lfloor vm/n \rfloor Z. \end{aligned}$$

and thus, since $t' = t$,

$$i - u = (v \bmod n/m - j \bmod n/m)n/k + \lfloor vm/n - jm/n \rfloor Z.$$

We have two cases.

- $\lfloor vm/n \rfloor = \lfloor jm/n \rfloor$ thus $v \bmod n/m - j \bmod n/m = v - j$ thus $0 \leq v - j \leq (i - u)k/n < 1$, that is $v = j$, and thus $i = u$ which leads to a contradiction.

- $\lfloor vm/n \rfloor \neq \lfloor jm/n \rfloor$ and thus

  $$|i - u| \geq (l + n^2/mk) - (n/m - 1)n/k \geq l + n/k > n/k$$

  which leads to a contradiction.

Let us assume as a hypothesis to be proved contradictory that s is temporally compromised. Thus there exists an edge, say $\eta = (\gamma_i^j, \gamma_u^v) \in \Delta$ such that for the unique tuples involving those tasks in s, say $(\gamma_u^v, p', t'), (\gamma_i^j, p, t)$, $t' < t + d(\eta, p, p') + 1$, where

$$\begin{aligned} t &= i \bmod n/k + (j \bmod n/m)n/k + \lfloor ik/n \rfloor Y + \lfloor jm/n \rfloor Z \\ t' &= u \bmod n/k + (v \bmod n/m)n/k + \lfloor uk/n \rfloor Y + \lfloor vm/n \rfloor Z \end{aligned}$$

We have three cases.

- $i = u$ and $v = j + 1$, in which case by the construction of s', $p = p'$ which implies $t' - t < 1$. However, $t' = i \bmod n/k + ((j + 1) \bmod n/m)n/k +$

$\lfloor ik/n \rfloor Y + \lfloor (j+1)m/n \rfloor Z$. Thus $t' - t = ((j+1) \bmod n/m - j \bmod n/m)n/k + (\lfloor (j+1)m/n \rfloor - \lfloor jm/n \rfloor)Z$ and since $Z \geq n^2/mk$, $t' - t \geq n/k \geq 1$, which leads to a contradiction.

- $j = v$ and $u = i + 1$ and $\lfloor ik/n \rfloor = \lfloor uk/n \rfloor$, that is $(u \bmod n/k) - (i \bmod n/k) = 1$ and by the construction of s, $p = p'$ which implies $t' - t < 1$. However, $t' = u \bmod n/k + (j \bmod n/m)n/k + \lfloor uk/n \rfloor Y + \lfloor jm/n \rfloor Z$. Thus $t' - t = 1$, which leads to a contradiction.

- $j = v$ and $u = i + 1$ and $\lfloor ik/n \rfloor = \lfloor uk/n \rfloor + 1$. Note that since the range of d is upper bounded by τ, $t' - t < \tau + 1$. However, $t' = u \bmod n/k + (j \bmod n/m)n/k + (\lfloor ik/n \rfloor + 1)Y + \lfloor jm/n \rfloor Z$ thus $t' - t = ((i+1) \bmod n/k - i \bmod n/k)n/m + Z$ and since $Z \geq \tau + n^2/mk$), $t' - t \geq n/m + \tau \geq 1 + \tau$ which leads to yet another contradiction.

Thus we conclude that in all cases s is not temporally compromised and since it is complete and not overbooked it is valid and the theorem is proved. □

Corollary 8.1 *Let $A = (P = \{p_0, \ldots, p_{k-1}\}, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a Fixed Delay UET scheduling model, where Λ is an $n \times n$ diamond dag and the range of d is $\{0, \tau\}$. Let s be a stripes schedule of A. s is valid for A.*

8.2.2 Lines

The lines approach is shown in Figure 8–3. Informally, each processor computes nodes a line at a time, that is, once it has computed a node in a line it continues computing nodes further up the line. When it finishes its line it waits until all k processors have started a line before starting the $k+1$'th line. Line j is assigned to processor $j \bmod k$. There are similar approaches to lines which can deliver better makespan, but the simplicity of lines leads to easy analysis.

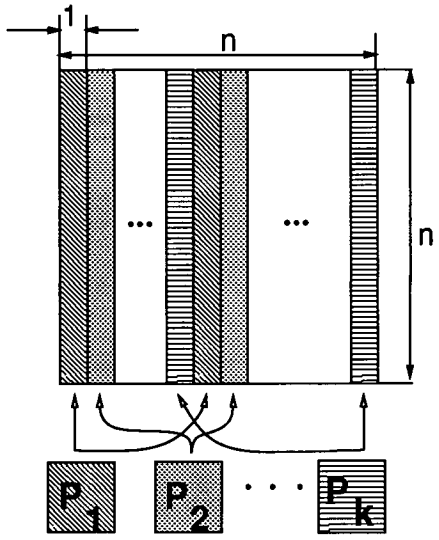


Figure 8–3: Lines Schedule in k Processors

Definition 8.3 Lines Schedule.
Let $A = (P = \{p_0, \ldots, p_{k-1}\}, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a *Fixed Delay* UET scheduling model, where Λ is an $n \times n$ diamond dag, the range of d is $\{0, \tau\}$ and $n \geq k(1+\tau)$.

For each $i = 0, \ldots, n-1$ let $p^i = p_{i \bmod k}$. For each $i = 0, \ldots, n-1$, for each $j = 1, \ldots, n-1$ let

$$t_i^j = (i \bmod k)(1+\tau) + j + n\lfloor i/k \rfloor$$

We define s, the *lines schedule* of A as follows.

$$s = \{(\gamma_i^j, p^i, t_i^j) : i = 0, \ldots, n-1, j = 0, \ldots, n-1\}$$

Theorem 8.2 *Let $A = (P = \{p_0, \ldots, p_{k-1}\}, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a Fixed Delay UET scheduling model, where Λ is an $n \times n$ diamond dag, the range of d is $\{0, \tau\}$. Let s be a lines schedule of A. s is valid for A.*

Proof
s is complete since, by construction, there is a tuple $(\gamma_i^j, p^i, t_i^j) \in s$ for each

$i = 0, \ldots, n-1, j = 0, \ldots, n-1$. Let us assume as a hypothesis to be proved contradictory that s is overbooked. Thus there exist two tuples, say $(\gamma_i^j, p, t) \in s$ and $(\gamma_l^m, p', t') \in s$ such that $p = p'$ and $t = t'$ and either one or both of $l \neq i$, $m \neq j$.

Note that by the construction of s, the following hold.

$$\begin{aligned} (i \bmod k)(1+\tau) + j + n\lfloor i/k \rfloor &= (l \bmod k)(1+\tau) + m + n\lfloor l/k \rfloor \\ i \bmod k &= l \bmod k \end{aligned}$$

Thus

$$j + n\lfloor i/k \rfloor = m + n\lfloor l/k \rfloor.$$

We have two cases $l = i$ or $|l - i| \geq k$.

- $l = i$, thus $n\lfloor i/k \rfloor = n\lfloor l/k \rfloor$, thus $m = j$ which leads to a contradiction.
- $|l - i| \geq k$, that is $|\lfloor i/k \rfloor - \lfloor l/k \rfloor| \geq 1$, thus $|j - m| \geq n$ which leads to a contradiction.

Let us assume as a hypothesis to be proved contradictory that s is temporally compromised. This would imply there existed an edge, say $\eta = (\gamma_i^j, \gamma_l^m) \in \Delta$ such that for the unique tuples involving those tasks in s, say $(\gamma_l^m, p', t'), (\gamma_i^j, p, t)$, $t' < t + d(\eta, p, p') + 1$, where

$$\begin{aligned} t &= (i \bmod k)(1+\tau) + j + n\lfloor i/k \rfloor \\ t' &= (l \bmod k)(1+\tau) + m + n\lfloor l/k \rfloor \end{aligned}$$

We have three cases.

- $i = l$ and $m = j + 1$, in which case by the construction of s, $p = p'$ which implies $t' - t < 1$. However, $t' = (i \bmod k)(1+\tau) + j + 1 + n\lfloor i/k \rfloor$, thus $t' - t = 1$ which leads to a contradiction.

- $j = m$ and $l = i+1$ and $\lfloor l/k \rfloor = \lfloor i/k \rfloor$. Note that since the range of d is upper bounded by τ, $t' - t < \tau + 1$. However, $t' = (i \bmod k+1)(1+\tau)+j+n\lfloor i/k \rfloor$ thus $t' - t = 1 + \tau$ which leads to another contradiction.

- $j = m$ and $l = i + 1$ and $\lfloor l/k \rfloor = \lfloor i/k \rfloor + 1$, thus $l \bmod k = 0$ and $i \bmod k = k - 1$. Note that since the range of d is upper bounded by τ, $t' - t < \tau + 1$. However, $t' = (1+\tau)+j+n(\lfloor i/k \rfloor + 1)$, thus $t' - t = n - (k-1)(1+\tau) = n - k(1+\tau)+1+\tau$, but by the definition of the stripes schedule $n \geq k(1+\tau)$ which leads to yet another contradiction.

Thus we conclude that in all cases s is not temporally compromised and since it is complete and not overbooked it is valid and the theorem is proved. □

8.2.3 Boxes

Papadimitriou and Ullman describe how a lower bound on event cost (for those partitions which evenly distribute work between processors) can be met by partitioning the diamond dag into boxes (Fig.8–4), each with a length of side of $n/\sqrt{k}$. Each box thus contains $n/\sqrt{k} \times n/\sqrt{k}$ nodes. We consider only the case where k has an integer square root which divides n exactly. The boxes schedule described below is not the only schedule to correspond to Papadimitriou and Ullman's boxes partition, but it has optimal makespan amongst them.

Definition 8.4 Boxes Schedule.
Let $A = (P = \{p_0, \ldots, p_{k-1}\}, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a *Fixed Delay* UET scheduling model, where Λ is an $n \times n$ diamond dag, the range of d is $\{0, \tau\}$, $\sqrt{k} \in Z_0^+$ and $n/\sqrt{k} \in Z_0^+$.[2]

[2]These two conditions serve merely to avoid notational excesses.

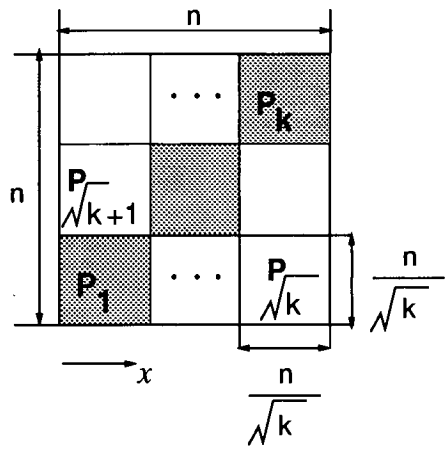


Figure 8–4: Boxes Schedule in k Processors

For each $i = 0, \ldots, n-1, j = 0, \ldots, n-1$ let us define the following

$$\begin{aligned} p_i^j &= p_{i \bmod n/\sqrt{k}+(j \bmod n/\sqrt{k})\sqrt{k}} \\ t_i^j &= j + \lfloor j\sqrt{k}/n \rfloor \tau + in/\sqrt{k} + \lfloor i\sqrt{k}/n \rfloor (1 - n/\sqrt{k} + \tau) \end{aligned}$$

We define s, the *boxes schedule* of A as follows.

$$s = \{(\gamma_i^j, p_i^j, t_i^j) : i = 0, \ldots, n-1, j = 1, \ldots, n-1\}.$$

Theorem 8.3 *Let $A = (P = \{p_0, \ldots, p_{k-1}\}, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a Fixed Delay UET scheduling model, where Λ is an $n \times n$ diamond dag, the range of d is $\{0, \tau\}$. Let s be a boxes schedule of A. s is valid for A.*

Proof

s is complete since, by construction, there is a tuple $(\gamma_i^j, p_i^j, t_i^j) \in s$ for each γ_i^j, $i = 0, \ldots, n-1, j = 0, \ldots, n-1$. Let us assume as a hypothesis to be proved contradictory that s is overbooked. Thus there exist two tuples, say $(\gamma_i^j, p, t), (\gamma_l^m, p', t') \in s$ such that $p = p'$ and $t = t'$ and one or both of $l \neq i$ and $m \neq j$.

Note that by the construction of s,

$$i \bmod n/\sqrt{k} + (j \bmod n/\sqrt{k})\sqrt{k} = l \bmod n/\sqrt{k} + (m \bmod n/\sqrt{k})\sqrt{k}.$$

Thus $i \bmod n/\sqrt{k} = l \bmod n/\sqrt{k}$ and $j \bmod n/\sqrt{k} = m \bmod n/\sqrt{k}$, which implies $\lfloor i\sqrt{k}/n \rfloor = \lfloor l\sqrt{k}/n \rfloor$, $\lfloor j\sqrt{k}/n \rfloor = \lfloor m\sqrt{k}/n \rfloor$, and $|j - m| < n/\sqrt{k}$. Note also, by the construction of s

$$\begin{aligned} t &= j + \lfloor j\sqrt{k}/n \rfloor \tau + in/\sqrt{k} + \lfloor i\sqrt{k}/n \rfloor (1 - n/\sqrt{k} + \tau) \\ t' &= m + \lfloor m\sqrt{k}/n \rfloor \tau + ln/\sqrt{k} + \lfloor l\sqrt{k}/n \rfloor (1 - n/\sqrt{k} + \tau) \end{aligned}$$

Thus since $t = t'$, $j - m = (l - i)n/\sqrt{k}$ which implies $l = i$ and thus $j = m$ which leads to a contradiction.

Let us assume as a hypothesis to be proved contradictory that s is temporally compromised. This would imply there existed an edge, say $\eta = (\gamma_i^j, \gamma_l^m) \in \Delta$ such that for the unique tuples involving those tasks in s, say $(\gamma_l^m, p', t'), (\gamma_i^j, p, t)$, $t' < t + d(\eta, p, p') + 1$, where

$$\begin{aligned} t &= j + \lfloor j\sqrt{k}/n \rfloor \tau + in/\sqrt{k} + \lfloor i\sqrt{k}/n \rfloor (1 - n/\sqrt{k} + \tau) \\ t' &= m + \lfloor m\sqrt{k}/n \rfloor \tau + ln/\sqrt{k} + \lfloor l\sqrt{k}/n \rfloor (1 - n/\sqrt{k} + \tau) \end{aligned}$$

We have four cases.

- $i = l$ and $m = j + 1$, and $\lfloor j\sqrt{k}/n \rfloor = \lfloor m\sqrt{k}/n \rfloor$ in which case by the construction of s, $p = p'$ which implies $t' - t < 1$. However, $t' = j + 1 + \lfloor j\sqrt{k}/n \rfloor \tau + in/\sqrt{k} + \lfloor i\sqrt{k}/n \rfloor (1 - n/\sqrt{k} + \tau)$, thus $t' - t = 1$ which leads to a contradiction.

- $j = m$ and $l = i + 1$, and $\lfloor i\sqrt{k}/n \rfloor = \lfloor l\sqrt{k}/n \rfloor$ in which case by the construction of s, $p = p'$ which implies $t' - t < 1$. However, $t' = j + \lfloor j\sqrt{k}/n \rfloor \tau + (i+1)n/\sqrt{k} + \lfloor i\sqrt{k}/n \rfloor (1 - n/\sqrt{k} + \tau)$, thus $t' - t = n/\sqrt{k} \geq 1$ which leads to another contradiction.

- $j = m$ and $l = i + 1$ and $\lfloor i\sqrt{k}/n \rfloor = \lfloor l\sqrt{k}/n \rfloor + 1$. Note that since the range of d is upper bounded by τ, $t' - t < \tau + 1$ However, $t' = j + \lfloor j\sqrt{k}/n \rfloor \tau + (i+1)n/\sqrt{k} + (\lfloor i\sqrt{k}/n \rfloor + 1)(1 - n/\sqrt{k} + \tau)$, thus $t' - t = 1 + \tau$ which leads to yet another contradiction.

- $i = l$ and $m = j + 1$ and $\lfloor m\sqrt{k}/n \rfloor = \lfloor j\sqrt{k}/n \rfloor + 1$. Note that since the range of d is upper bounded by τ, $t' - t < \tau + 1$ However, $t' = j + 1 + (\lfloor j\sqrt{k}/n \rfloor + 1)\tau + in/\sqrt{k} + \lfloor i\sqrt{k}/n \rfloor (1 - n/\sqrt{k} + \tau)$, thus $t' - t = 1 + \tau$ which leads to our final contradiction.

Thus we conclude that in all cases s is not temporally compromised and since it is complete and not overbooked it is valid and the theorem is proved. □

8.3 The Predictions of the Models

Let $A = (P = \{p_0, \ldots, p_{k-1}\}, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a *Fixed Delay* UET scheduling model, where Λ is an $n \times n$ diamond dag and the range of d is $\{0, \tau\}$. Let $s_A^{stripes}$ be a stripe schedule of A. Let s_A^{lines} be a lines schedule of A. Let s_A^{boxes} be a boxes schedule of A.

We can obtain the makespans of the schedules by considering the execution of γ_{n-1}^{n-1} which depends upon every other task. Since all tasks are executed exactly once in the schedules we have defined, it is the last task to be executed, and by adding 1 to its execution time we obtain the makespan of the schedule.

$$\begin{aligned}
\mathcal{M}(s_A^{stripes}) &= (n-1) \bmod n/k + ((n-1) \bmod n/n)n/k + \\
&\quad \lceil (n-1)k/n \rceil Y + \lceil (n-1)k/n \rceil Z + 1 \\
&= n/k - 1 + (k-1)(\tau + n/k) + (n-1)(n/k) + 1 \\
&= n^2/k + (k-1)n/k + (k-1)\tau.
\end{aligned}$$

$$\begin{aligned}
\mathcal{M}(s_A^{lines}) &= ((n-1) \bmod k)(1+\tau) + n - 1 + n\lfloor (n-1)/k \rfloor + 1 \\
&= (k-1)(1+\tau) + n - 1 + n^2/k - n + 1 \\
&= n^2/k + (k-1) + (k-1)\tau. \\
\mathcal{M}(s_A^{boxes}) &= n - 1 + \lfloor (n-1)\sqrt{k}/n \rfloor (2\tau + 1 - n/\sqrt{k}) + (n-1)n/\sqrt{k} \\
&= n^2/\sqrt{k} + \sqrt{k} - 1 + 2(\sqrt{k}-1)\tau.
\end{aligned}$$

Note $n/k \geq 1$, thus for all $\tau \geq 1$, $\mathcal{M}(s_A^{lines}) \geq \mathcal{M}(s_A^{stripes})$.

We observe that $\mathcal{M}(s_A^{boxes})$ has a smaller multiplicative factor of τ than either $\mathcal{M}(s_A^{lines})$ or $\mathcal{M}(s_A^{stripes})$. Note that $\max_{t=0}^{\mathcal{M}(s)-1} \mathcal{A}(s_A^{boxes}, t) = 2\sqrt{k}$. Both the stripes and the lines schedules would have an identical multiplicative factor if they were restricted to $2\sqrt{k}$ processors.

Let $B = (P = \{p_0, \ldots, p_{k-1}\}, \Lambda = (\Gamma, \Delta), f, d)$ be an instance of a *No Delay* UET scheduling model, where Λ is an $n \times n$ diamond dag. Let $s_B^{stripes}$ be a stripes schedule of B. Let s_B^{lines} be a lines schedule of B. Let s_B^{boxes} be a boxes schedule of B.

We have the following values for makespan.

$$\begin{aligned}
\mathcal{M}(s_B^{stripes}) &= n^2/k + (k-1)n/k. \\
\mathcal{M}(s_B^{lines}) &= n^2/k + (k-1). \\
\mathcal{M}(s_B^{boxes}) &= n^2/\sqrt{k} + \sqrt{k} - 1.
\end{aligned}$$

Note that each of our three schedules of B contains a single tuple for each task in Γ. It thus has a unique tuple dependence graph,of which we shall refer to the subset containing all *interprocessor dependences* as, for example $D_B^{stripes}$. For all $k > 1$ we have the following values for event cost.

$$\begin{aligned}
\mathcal{E}_{D_B^{stripes}} &= n(k-1). \\
\mathcal{E}_{D_B^{lines}} &= n^2 - n. \\
\mathcal{E}_{D_B^{boxes}} &= 2n(\sqrt{k}-1).
\end{aligned}$$

Note $n \geq k$, thus $\mathcal{E}_{D_B^{lines}} \geq \mathcal{E}_{D_B^{stripes}}$.

For all $k > 1$ we have the following values for event cost.

$$\begin{aligned} C_{D_B^{stripes}} &= k - 1. \\ C_{D_B^{lines}} &= n - 1. \\ C_{D_B^{boxes}} &= 2(\sqrt{k} - 1). \end{aligned}$$

Note $n \geq k$, thus $C_{D_B^{lines}} \geq C_{D_B^{stripes}}$.

It is clear that although the lines schedule fares particularly badly in having a larger event cost and chain cost than either of the other schedules, it performs the best out of the three schedules for any value of $n/k > 1$, and for any value of τ if it is restricted to a reasonable number of processors. The programmer who is faced with the problem of explicitly scheduling a computation corresponding to the diamond dag, should therefore make different qualitative decisions on the schedule, and write different programs, according to which cost model best approximates his or her architecture.

8.4 The Claud Stripes Schedule

In this section we consider two versions of the Claud model: the *Fixed Delay No Cost* Claud model and the *Fixed Delay Fixed Cost* Claud model. We consider the performance of schedules corresponding to Papadimitriou and Ullman's stripes partitions, with a restricted class of message partitions which are indexed by a single positive integer parameter m, where km is the number of fixed length messages into which the set of interprocessor tuple dependences is partitioned when the diamond dag is scheduled on k processors.

As the *Fixed Delay No Cost* Claud model is a special case of the *Fixed Delay* Claud model we present analysis only for the *Fixed Delay* Claud model. The

Stripes Claud Schedule defined below is a *holey stripes* schedule with extra send and receive tasks scheduled to occur during the holes. The execution times of the send and receive tasks add up to form the hole width, and the hole count corresponds to the number of messages that are used to send the precedence arcs from tasks that have been mapped to a given processor. The schedule is shown diagrammatically in Figure 8–5.

Definition 8.5 Stripes Claud Schedule.
Let m be a positive integer. Let $A = (P = \{p_0, \ldots, p_{k-1}\}, \Lambda = (\Gamma', \Delta), f', d, \sigma, \rho)$ be an instance of a *Fixed Delay* Claud model, where Λ is an $n \times n$ diamond dag, the range of d is $\{0, \tau\}$, $n/k \in Z_0^+$, and $f'(\sigma) = \lambda_\sigma$, $f'(\rho) = \lambda_\rho$.

Let f be the restriction of f' to the domain Γ. Let $B = (P, (\Gamma' - \{\sigma, \rho\}, \Delta), f, d)$. Note that B is an instance of a fixed delay *scheduling* model. Let s' be a holey stripes schedule of B with hole width $l = l_s + l_r$ and hole count m such that $n/m \in Z_0^+$. Let $Y = (l + n/m\tau \dot{+} n^2/mk)$. Let $Z = (l + n^2/mk)$. For each $x = 1, \ldots, k-1, y = 1, \ldots, m$ let us define the following

$$v_x^y = Yx + Zy - (l + n/m\tau)$$

$$r_x^y = Yx + Zy - l_r$$

s, the stripes Claud schedule of A with m messages per processor, is defined as follows.

$$\begin{aligned} s \;=\; & s' \cup \{(\sigma, p_{x-1}, v_x^y) : x = 1, \ldots, k-1, y = 1, \ldots, m\} \cup \\ & \{(\rho, p_x, r_x^y) : x = 1, \ldots, k-1, y = 1, \ldots, m\} \end{aligned}$$

The above schedule specifies the timing of communication events, but not the exact set of precedence arcs which are sent in each message. We send arcs in messages of equal length. The exact message partition is defined below.

Definition 8.6 Equal Message Partition for Stripes.
Let $A = (P = \{p_0, \ldots, p_{k-1}\}, \Lambda = (\Gamma, \Delta), f', d, \sigma, \rho)$ be an instance of a *Fixed Delay*

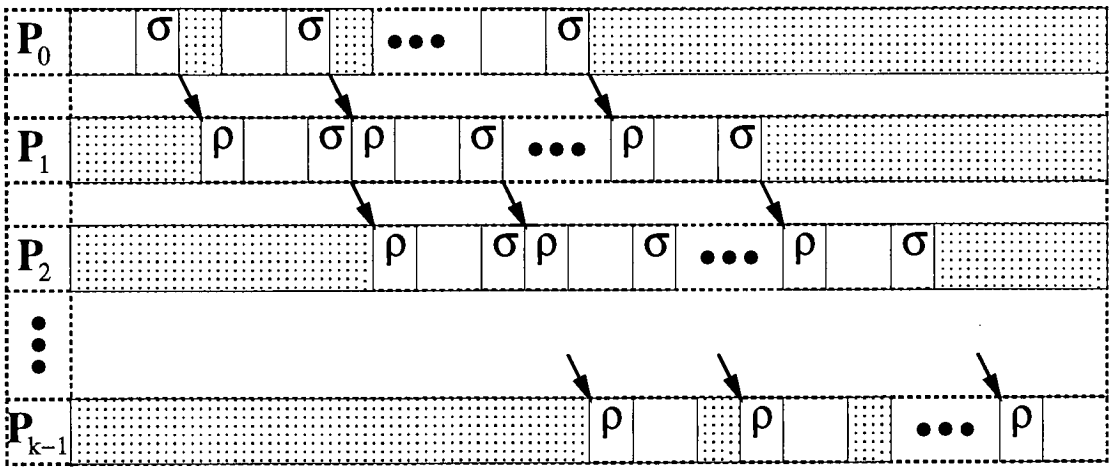


Figure 8–5: Claud Stripes Schedule of the $n \times n$ Diamond dag

Claud model, where Λ is an $n \times n$ diamond dag and $n/k \in Z_0^+$. Let s be a stripes Claud schedule of A. Let s' be the set of tuples instancing tasks in the diamond itself (rather than send and receive tasks). More formally, let

$$\begin{aligned} s' &= X(s, \Gamma, P, Z_0^+) \\ &= \{(\gamma_i^j, p_i^j, t_i^j) : i = 0, \ldots, n-1, j = 0, \ldots, n-1\}. \end{aligned}$$

We define M, the equal message partition of s with m messages per processor as follows. Let

$$M = \{M_x^y : x = 1, \ldots, k-1, y = 1, \ldots, m\}$$

where, for $x = 1, \ldots, k-1$, for $y = 1, \ldots, m$, M_x^y is the yth message sent to processor x, and

$$M_x^y = \{((\gamma_{i-1}^j, p_{i-1}^j, t_{i-1}^j), (\gamma_i^j, p_i^j, t_i^j)) : i = xn/k, j = (x-1)n/m, \ldots, xn/m - 1\}.$$

In order to later show the validity of the Claud schedule we need to show that it includes a set of communication events which can support a message partition of all the implied interprocessor communication. We now show that the schedule does indeed support the equal message partition described above.

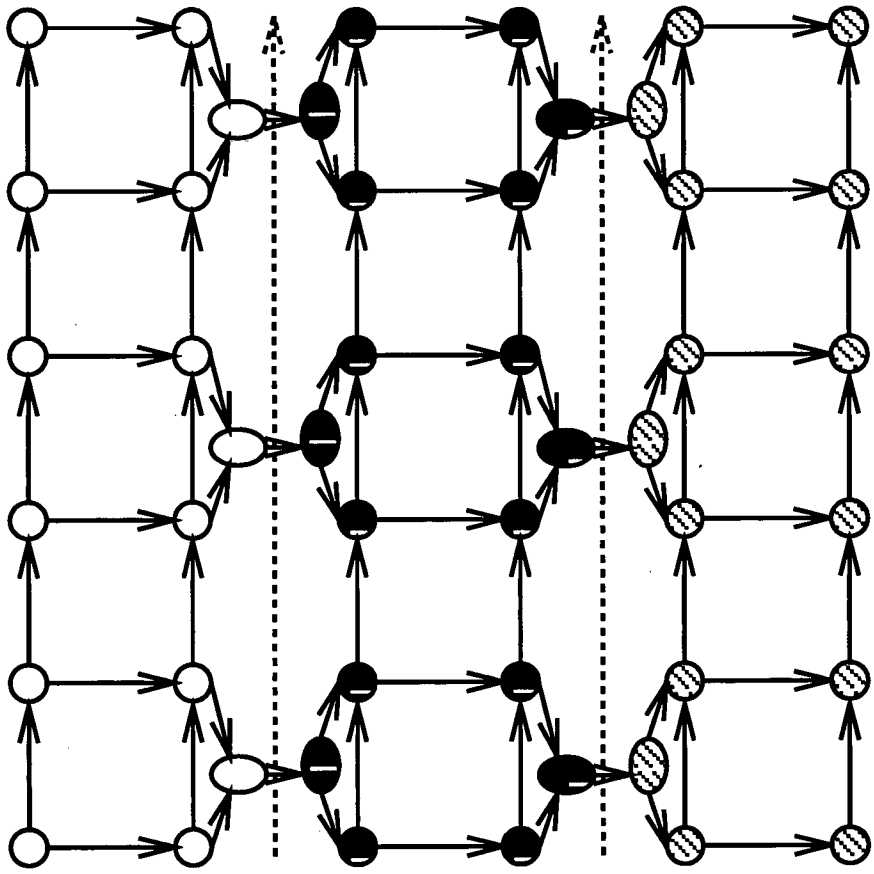


Figure 8–6: Equal Message Partition of the Stripes Claud Schedule of the 6×6 Diamond with $n/m = 2$ and $n/k = 2$ Showing Corresponding Send and Receive Events.

Theorem 8.4 *Let m be a positive integer. Let $A = (P = \{p_0, \ldots, p_{k-1}\}, \Lambda = (\Gamma, \Delta), f', d, \sigma, \rho)$ be an instance of a Fixed Delay Claud model, where Λ is an $n \times n$ diamond dag and $n/k \in Z_0^+$. Let s be a stripes Claud schedule of A with m messages. Let $M = \{M_x^y : x = 1, \ldots, k-1, j = 1, \ldots, m\}$ be an equal message partition for s. s supports M.*

Proof

Let $f'(\sigma) = l_s$ and $f'(\rho) = l_r$. Let $l = l_r + l_s$. Let D be the unique tuple dependence graph for s. Let C be the set of all interprocessor tuple dependences in D. Note that M partitions D. Let $Y = (l + n/m\tau + n^2/mk)$. Let $Z = (l + n^2/mk)$. Note the following properties of M_x^y, $x = 1, \ldots k-1, y = 1, \ldots, m$:

$$\begin{aligned} |M_x^y| &= n/m. \\ \mathcal{R}(M_x^y) &= t_{yn/k-1}^{(x+1)n/m-1} = xY + yZ - (l + n/m\tau + 1). \\ \mathcal{D}(M_x^y) &= t_{yn/k}^{xn/m} = xY + yZ. \\ \mathcal{L}(M_x^y) &= l + n/m\tau + 1. \end{aligned}$$

That is for each M_x^y, $x = 1, \ldots k-1, y = 1, \ldots, m$, $\mathcal{L}(M_x^y) > n/m\tau = |M_x^y|\tau$, that is, by Definition 7.2, M_x^y is valid for D. Let us define the following:

$$\begin{aligned} S &= X(s, \{\sigma\}, P, Z_0^+) = \{(\sigma, p_x, v_x^y), x = 1, \ldots k-1, y = 1, \ldots, m\}. \\ R &= X(s, \{\rho\}, P, Z_0^+) = \{(\rho, p_{x-1}, r_x^y), x = 1, \ldots k-1, y = 1, \ldots, m\}. \end{aligned}$$

Let $\mathcal{F}_s : M \to S$ and $\mathcal{F}_r : M \to R$ be defined such that for each M_x^y, $x = 1, \ldots k-1, y = 1, \ldots, m$, $\mathcal{F}_s(M_x^y) = (\sigma, p_{x-1}, v_x^y)$ and $\mathcal{F}_r(M_x^y) = (\rho, p_x, r_x^y)$. Note that for each $x = 1, \ldots k-1, y = 1, \ldots, m$,

$$\begin{aligned} \mathcal{R}(M_x^y) &= xY + yZ - (l + n/m\tau + 1) < v_{x-1}^y. \\ \mathcal{D}(M_x^y) &= xY + yZ > r_x^y. \end{aligned}$$

Thus,by Definition 7.6, s supports M. □

Now we are in a position to prove the validity of our stripes Claud schedule.

Theorem 8.5 *Let m be a positive integer. Let $A = (P = \{p_0, \ldots, p_{k-1}\}, \Lambda = (\Gamma, \Delta), f', d, \sigma, \rho)$ be an instance of a Fixed Delay Claud model, where Λ is an $n \times n$ diamond dag , $n/k \in Z_0^+$ and $n/m \in Z_0^+$. Let s be a stripes Claud schedule of A with m messages. s is valid.*

Proof

Let $f'(\sigma) = l_s$, $f'(\rho) = l_r$. Let f be the restriction of f' to the range Γ. Let $s' = X(s, \Gamma, P, Z_0^+)$. Note s' is a holey stripes schedule of (P, Λ, f, d) with hole count m and hole size $l_s + l_r$, thus s' is valid.

Let D be the unique tuple dependence graph for s. Let C be the set of all interprocessor tuple dependences in D. Let $M = \{M_x^y : x = 1, \ldots, k-1, y = 1, \ldots, m-1\}$ be an equal message partition for s. M partitions C and s supports M.

Finally consider whether s is overbooked. Note s' is valid and therefore not overbooked and thus no two tasks in Γ can share the same time step. Thus we have five cases.

- There exist two tuples $(\sigma, p_{x-1}, v_x^y), (\sigma, p_{x-1}, v_x^{y'}) \in \Gamma'$ such that $0 < v_x^y - v_x^{y'} < l_s$. Now $v_x^y = xY + yZ - (l + n\tau/m)$ and $v_x^{y'} = xY + Zy' - (l + n\tau/m)$ and thus $v_x^y - v_x^{y'} = Z(y - y')$ which, since $Z \geq l_s$ leads to a contradiction.

- There exist two tuples $(\rho, p_x, r_x^y), (\rho, p_x, r_x^{y'}) \in \Gamma'$ such that $0 < r_x^y - r_x^{y'} < l_r$. Now $r_x^y = xY + yZ - l_r$ and $r_x^{y'} = xY + yZ - l_r$ and thus $r_x^y - r_x^{y'} = Z(y - y')$ which, since $Z \geq l_r$ leads to a contradiction.

- There exists two tuples $(\sigma, p_x, v_{x+1}^y), (\rho, p_x, r_x^{y'}) \in \Gamma'$ such that either $r_x^{y'} \leq v_{x+1}^y < r_x^{y'} + l_r$ or $v_{x-1}^y \leq r_x^{y'} < v_{x-1}^y + l_s$. Thus $-l_s < v_{x-1}^y - r_x^{y'} < l_r$. However, $v_{x-1}^y = (x-1)Y + yZ - (l + n\tau/m)$ and $r_x^{y'} = xY + y'Z - l_r$ and thus $v_x^y - r_{x-1}^{y'} = Z(y - y' + 1) - l_s$. We have two subcases

- $y - y' \leq -1$ thus $v_x^y - r_{x-1}^{y'} \leq -l_s$ which leads to a contradiction.
- $y - y' > -1$ thus $v_x^y - r_{x-1}^{y'} \geq l - l_s = l_r$ which leads to a contradiction.

- There exist two tuples $(\rho, p_x, r_x^y), (\gamma_i^j, p_x, t_i^j) \in \Gamma'$ such that $0 < v_x^y - t_i^j < l_s$. Now $r_x^y = xY + yZ - l_r$

$$t_i^j = i \bmod n/k + (j \bmod n/m)n/k + \lfloor ik/n \rfloor Y + \lfloor jm/n \rfloor Z.$$

Thus, since $x = \lfloor ik/n \rfloor$

$$r_x^y - t_i^j = yZ - i \bmod n/k + (j \bmod n/m)n/k + \lfloor jm/n \rfloor Z - l_r,$$

which implies

$$(y - \lfloor jm/n \rfloor)Z - l_r \leq r_x^y - t_i^j < (y - \lfloor jm/n \rfloor + 1)Z - l - l_r$$

We have two subcases

- $y - \lfloor jm/n \rfloor > 0$ thus $r_x^y - t_i^j \geq Z - l_r \geq l_s$ which leads to a contradiction.
- $y - \lfloor jm/n \rfloor \leq 0$ thus $r_x^y - t_i^j < -l - l_r \leq 0$ which leads to a contradiction.

- There exist two tuples $(\sigma, p_x, v_{x-1}^y), (\gamma_i^j, p_x, t_i^j) \in \Gamma'$ such that $0 < v_{x-1}^y - t_i^j < l_s$. Now $v_{x-1}^y = (x-1)Y + yZ - (l + n\tau/m)$ and

$$t_i^j = i \bmod n/k + (j \bmod n/m)n/k + \lfloor ik/n \rfloor Y + \lfloor jm/n \rfloor Z.$$

Thus, since $x = \lfloor ik/n \rfloor$

$$v_x^y - t_i^j = yZ - i \bmod n/k + (j \bmod n/m)n/k + \lfloor jm/n \rfloor Z + Y - (l + n\tau/m),$$

which, since $Y - (l + n\tau/m) = Z$ implies

$$(y + 1 - \lfloor jm/n \rfloor + 1)Z \leq v_x^y - t_i^j < (y + 1 - \lfloor jm/n \rfloor)Z - l$$

We have two subcases

- $y + 1 - \lfloor jm/n \rfloor > 0$ thus $v_x^y - t_i^j \geq Z - l_r \geq l_s$ which leads to a contradiction.
- $y + 1 - \lfloor jm/n \rfloor \leq 0$ thus $v_x^y - t_i^j < -l - l_r \leq 0$ which leads to a contradiction.

□

8.4.1 Predicted Performance for the Claud Models

In this section we consider the performance of a stripes Claud schedule. Again we can obtain the makespans of the schedules by considering the execution of γ_{n-1}^{n-1} which depends upon every other task, and by adding 1 to its execution time we obtain the makespan of the schedule.

Let m be a positive integer. Let $A = (P = \{p_0, \ldots, p_{k-1}\}, \Lambda = (\Gamma, \Delta), f', d, \sigma, \rho)$ be an instance of a *Fixed Delay* Claud model, where Λ is an $n \times n$ diamond dag , $n/k \in Z_0^+$ and $n/m \in Z_0^+$. Let s be a stripes Claud schedule of A with m messages. Let $X = (l + n/m\tau + n^2/mk)$. Let $Y = (l + n^2/mk)$. Let $l = f'(\sigma) + f'(\rho)$.

$$\mathcal{M}(s) = (k-1)(l + n/m\tau + n^2/mk) + m(l + n^2/mk) - l.$$

In order to model fixed (but not unit) execution time computations we introduce a node computation time ω and consider l and τ to be normalised with respect to it, that is $l = \hat{l}/\omega$ and $\tau = \hat{\tau}/\omega$, where $\hat{l}$ and $\hat{\tau}$ are parameters of our multicomputer architecture which may be measured as in Chapter 9 below.

In Figure (8–7) $\mathcal{M}(s)$ is shown as a three dimensional surface for $\hat{l}, \hat{\tau}, \omega$ as derived in a section 9.2.1. There are points in k and m where $\mathcal{M}(s)$ has a minimum. A slow linear increment of $\mathcal{M}(s)$ is observed when k or m are above the minimal point and a rapid nonlinear growth takes place for values of k and m below the minimal point.

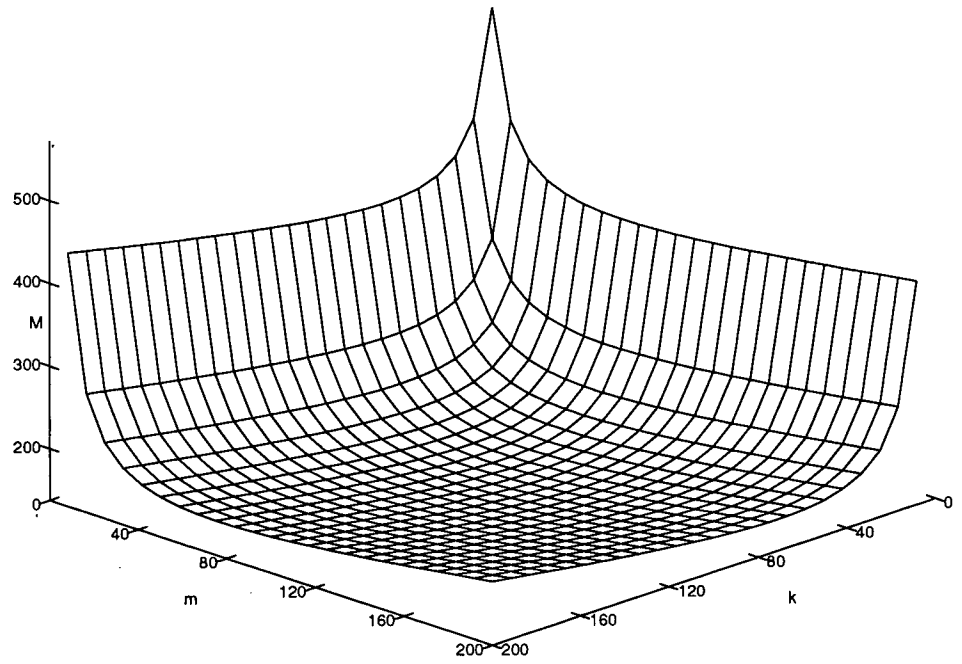


Figure 8–7: Makespan M as a Function of two Variables k and m, for $n = 240, \hat{\lambda} = 475, \hat{\tau} = 0.775, \omega = 43.30$.

If we take the liberty of considering $\mathcal{M}(s)$ as a continuous function of continuous variables, we may find minima by differentiation. When k is fixed we may find its extrema by solving the equation

$$\frac{\partial \mathcal{M}(s)}{\partial m} = 0.$$

we note the second derivative of $\mathcal{M}(s)$ by m is positive in the interval $(0, \infty)$ and thus the extremum is a minimum . An analogous consideration is true when we consider $\mathcal{M}(s)$ as a function of k with fixed m.

Let s_{opt} be a schedule of optimal makespan. For the purposes of our analysis we shall assume that $\frac{(k-1)}{k}$ is close to 1, so the approximate formula for $\mathcal{M}(s_{opt})$ is

$$\mathcal{M}_{opt} = \frac{n}{\sqrt{l}}\sqrt{1 + \tau\frac{k}{n}}(1 + O(\frac{1}{n})).$$

Chapter 9

Validation of the models

In this chapter we test the quantitative predictive power of the Claud model on a computation which can be represented as the diamond dag. This we do by performing the computation in a manner corresponding to the stripes schedule outlined in Chapter 8, on different numbers of processors and with different message partitions. We compare the measured performance of the computation with that predicted by the model. We measure the parameters of the models directly from benchmarking the computation. Also, by using regression analysis, we attempt to find sets of parameters which optimally fit the data points. Finally we discuss the predictive power of our models in terms of features of our multicomputer and our software.

9.1 A Numerical Application

Given a set of partial differential equations that describe a (continuous) physical process, a solution can be obtained using a discrete approximation.

The discretisation procedures we consider here are such that the unknowns are coupled together within a regular $n \times n$ rectangular mesh. This can be achieved by the 5-point finite difference discretisation (e.g. see [Wachpress 1966]) which leads to sparse positive definite linear systems.

In order to express a linear system with a matrix form $Ax = b$ of size $n \times n$, there must exist a correspondence between equations and unknowns, established by choosing an ordering; we consider here the natural or lexicographical ordering (e.g. see [Hageman and Young 1981] or [Beauwens 1989]).

The method we consider to solve the matrix equation is an iterative method where the preconditioning matrix B is obtained from an incomplete symmetric factorisation of A without fill-in (see [Beauwens 1989] for full details). B is of the form LU (several normalisations are possible) where U is upper triangular

and L lower triangular and the non-zero element pattern of $(L+U)$ is the same as the non-zero element pattern of A.

Whatever the iterative method used it will require at each iteration (at least) the inversion of B. This means solving a matrix equation of the form $By = k$ which is performed in two steps:

$Lz = k$, yielding z (this is the *forward solve*) and then $Ux = z$ yielding x (this is the *backward solve*).

These two equations being similar, we limit our discussion to the first one within which the general expression of the unknown $z[i,j]$ at the point $[i,j]$ of the mesh is of the form:

$$k[i,j] = q[i,j]z[i,j] + q[i-1,j]z[i-1,j] + q[i,j-1]z[i,j-1]$$

From that expression it can be seen clearly that the computation of each unknown of the mesh requires the results of the south and west neighbours.

To represent this computation in the form of a dag, we have one node for each mesh element and arcs entering the node from nodes corresponding to these neighbouring south and west mesh elements (where such neighbours exist). The result is a diamond dag.

The parallel execution of the computation consists of the execution of the computation of different unknowns on different processors, and the sending of messages between processors. The unknowns were executed on processors to which corresponding tasks had been mapped in the stripes Claud schedule, and messages were sent containing data corresponding to edges as in the equal message partition. We are unable to specify the timing of computations on each processor (indeed to do so would mean that we were not testing the predictive power of the model). Instead our program specified an order of task execution and message send and receive calls which was consistent with the stripes Claud schedule.

9.2 A Validation Experiment

Given the expression derived above for Makespan in the *fixed cost fixed delay* Claud model, we wish to compare the predictive power of the *no cost fixed delay* Claud model and the *fixed cost fixed delay* Claud model for a computation on k processors corresponding to the the $n \times n$ diamond dag and sending m equal sized messages per processor.

The computation associated with a node in our example involves three additions, two multiplications and a division, all at double precision floating point. It also involves computations to index into arrays during the loop. The computation was implemented in C on a on a Meiko Computing Surface [Meiko 1992] containing 20MHz Inmos T800 transputers [inmos 1988] with 20 Mbit/s links through Meiko switch chips.

In order to match the computation to the Claud stripes schedule, it was coded as an outer loop over m blocks that make up a stripe, a centre loop over the n/m strips of a block, and an inner loop over n/k nodes of a strip. Each stripe was assigned to a different processor. Processors assigned adjacent stripes were connected by a single physical transputer link. The arrays were arranged so that the last elements of each strip were contiguous in memory. These elements were sent in messages of length n/m using communication calls are put in the outer loop.

The communication was effected by `csn` calls in Meiko's CS-tools version 1.19 Messages were sent by non-blocking[1] `csn_txnb` calls immediately preceded (in all but the first from each processor) by a `csn_test` test for completion.. A

[1]This corresponds to the *asynchronous* call of some other message passing systems. See [Meiko 1992] for an explanation.

final `csn_test` was made by each processor. Messages were received by blocking `csn_rx` calls. Each of these send and receive calls was surrounded by a small "wrapper" to test its return status.

9.2.1 Deriving Parameters of the Model

The computation is parameterised by n, the index of the diamond, ω, the node execution time, $\hat{l} = \hat{l_s} + \hat{l_r}$, the time required to set up message transfers, k, the number of processors used, m the number of messages each processor sends and $\hat{\tau}$ the time required to send a unit length message between processors. m, n and k are well-defined. $\hat{l_s}, \hat{l_r}, \omega$ and $\hat{\tau}$ were measured as below.

Message send and receive overheads were measured by timing a large loop, and measuring the extension to that time when messages were being repeatedly sent to or received from adjacent processors during the loop. The values include a small overhead (about $25\mu s$) associated with the "wrappers".

$$\hat{l_s} = 270\mu s$$

$$\hat{l_r} = 205\mu s$$

$\hat{\tau}$ was considered to be the reciprocal of the bandwidth available between adjacent processors for asymptotically large messages. This was computed by measuring the time taken to echo a one megabyte message between adjacent processors and subtracting the time taken to echo a one byte message. The result was divided by $2 \times (2^{20} - 1)$. This gave a value which (owing to vagaries of the Meiko hardware) depended slightly upon the pair of physical processors to which the processes were mapped.

$$\hat{\tau} = 0.775\mu s/byte$$

It is possible to "optimise" the computation by using, at all levels of the loop, pointers instead of indices. If this is done then the computation time for a node is strongly dependent upon the block size and stripe (or strip) width. This effect is taken into account in [Manneback *et al.* 1992], but we chose to analyse results for a naïve version of the software where arrays were always accessed with explicit array indexing on loop counters. Fortunately, from the point of view of our experiments, the Meiko compiler chose not to optimise this aspect of our code and the ω values are relatively consistent for different block sizes and stripe widths. As a point of interest, the asymptotic "optimised" value is $13.96\mu s$.

$$\omega = 43.30\mu s$$

9.2.2 Predicted Versus Achieved Performance

Figures 9–1 and 9–2 show two cross-sections through the basket shape shown in Figure 8–7 for $n = 240, \hat{l} = 475, \hat{\tau} = 0.775, \omega = 43.30$ for the performance of the numerical application on a Meiko Computing Surface, with parameters as derived in Section 9.2.1. Figure 9–1 shows the performance for $m = 80$ and where k is a factor of 240 in the range 1 to 80 inclusive. Figure 9–2 shows the performance for $k = 80$ and where k is a factor of 240 in the range 1 to 240 inclusive. The data points indicated by crosses were measured times of execution. The predicted performance is also shown. In each case, the lower of the two lines corresponds to the performance prediction of the *Fixed Cost* Claud model, and the upper line corresponds to the *Fixed Cost Fixed Delay* Claud model. It is clear that the *Fixed Cost Fixed Delay* Claud model is a good predictor of the performance of our computation. Indeed, in no case do the prediction and the data points vary by more than 5%.

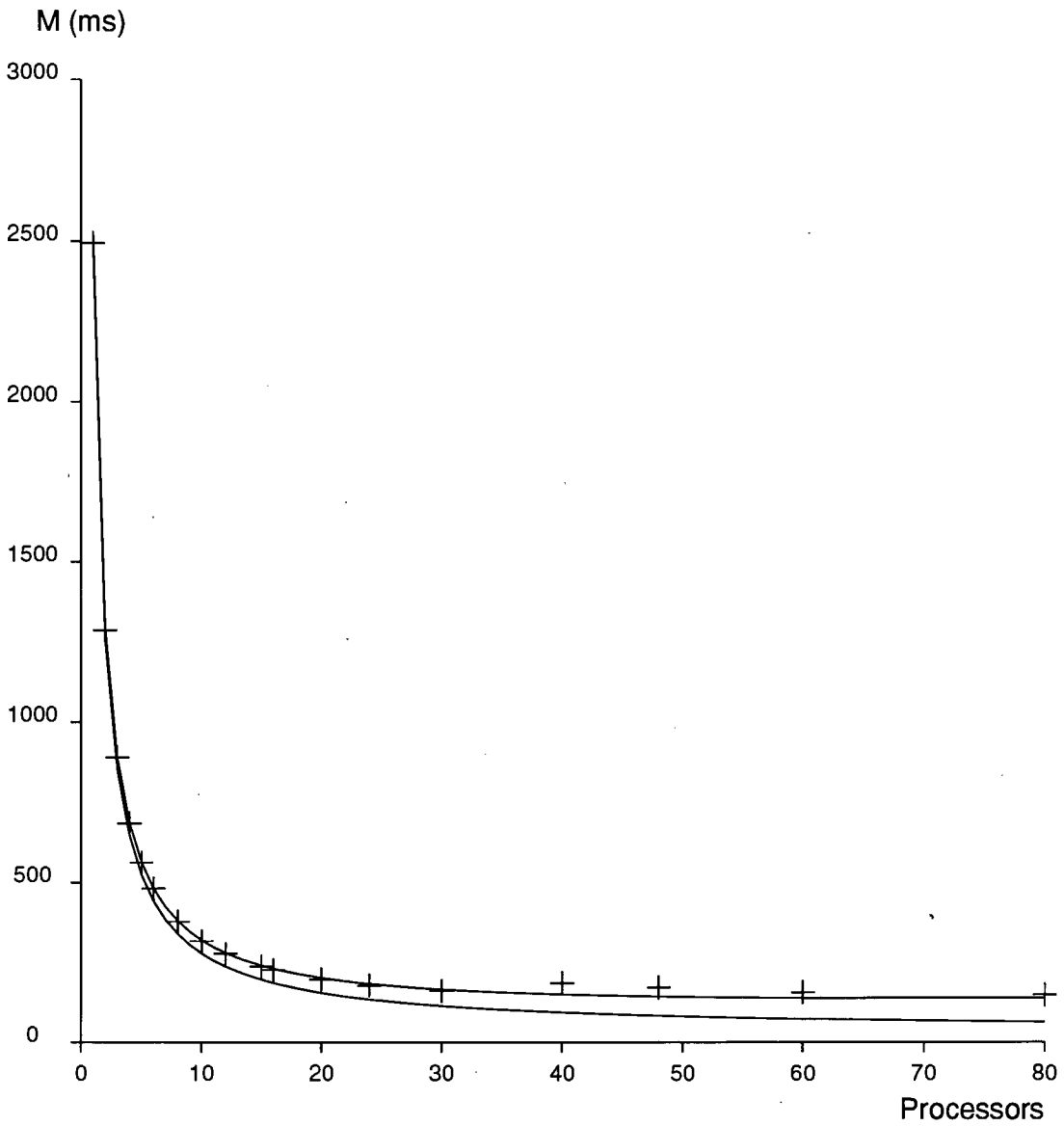


Figure 9–1: Performance with Constant Message Size: Observed (points) *vs* Predicted (line)

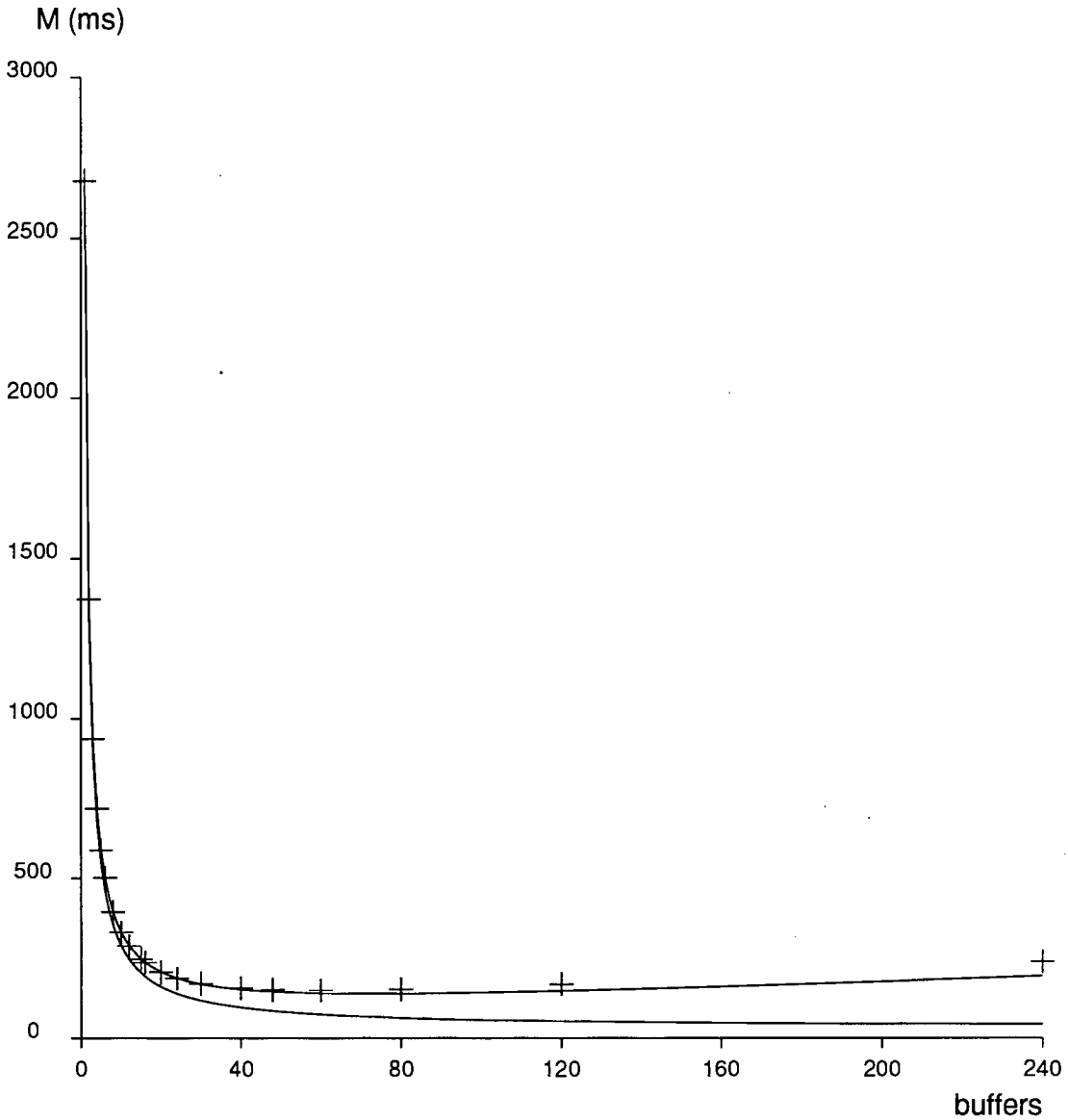


Figure 9–2: Performance with Constant Number of Processors: Observed (points) *vs* Predicted (line)

9.2.3 Fitting the Models to the Observed Data

The analysis still, however, begs the question as to whether it is possible to predict the performance of the computation equally as effectively with a *No Cost Fixed Delay* Claud model. It is clear that we had the *Fixed Cost Fixed Delay* Claud model in mind when we were deriving the parameters of the architecture, and therefore it is not simply sufficient to use in the *No Cost Fixed Delay* Claud model the value of $\hat{\tau}$ we derived earlier. A more rigourous statistical treatment is required.

We choose to analyse the respective efficacies of the models by deriving best-fit parameters by regression analysis and then analysing the goodness of fit of the curves that are predicted, in terms of its statistical significance by analysis of variance. In the case of the *No Cost Fixed Delay* Claud model we use regression analysis to find the single parameter $\hat{\tau}$. In the case of the *Fixed Cost Fixed Delay* Claud model we use the analysis to find both $\hat{l}$ and $\hat{\tau}$. Then, for the *Fixed Cost Fixed Delay* Claud model we consider the deviations between the regression parameters and the measured parameters, in terms of confidence limits that we derive for the regression parameters.

There is an assumption in the analysis that the deviation of the experimental values from the values predicted by the expression is normally distributed. It is less than clear how one could set about verifying this assumption, but it is a standard assumption in this type of analysis.

Numerical solution of the regression coefficients for the *Fixed Cost Fixed Delay* Claud model gives

$$\hat{l} = 531\mu s, \quad \hat{\tau} = 0.787\mu s/byte$$

which values are remarkably close to those derived by our independent benchmarking of the multicomputer.

Analysis of variance gives a variance ratio which, according to tables of the significance, indicates that the probability that a fit as good as that which is predicted by a Claud model (with the regression parameters) could have arisen by chance is very much less than 0.1%.

Numerical solution of the regression coefficients for the *No Cost Fixed Delay* Claud model gives

$$\hat{\tau} = 2.02\mu s/byte$$

Analysis of variance gives a variance ratio which again indicates that the probability that a fit as good as that which is predicted by a Claud model (with the regression parameters) could have arisen by chance is very much less than 0.1%.

The curves derived from parameters predicted by the regression analyses are shown in Figures 9–3 and 9–4.

The *Fixed Cost Fixed Delay* Claud model correctly predicts the increase in computation time seen with large numbers of processors and with large numbers of messages, whereas in The *No Cost Fixed Delay* Claud model the optimal number of processors is one for each line. In addition, the optimal message size is 1 since if the transfer of the data associated with a precedence arc is delayed by putting it into a message with another arc, this can never cause a reduction in the computation time of the schedule.

9.3 Parameter Dependence in the Claud model

Given that we have some confidence in the predictive power of *Fixed Cost Fixed Delay* Claud model for the Diamond dag computation, it is interesting to consider how the performance of the computation would be improved by changes to hardware and software. For a fixed k, ω and n, we can consider

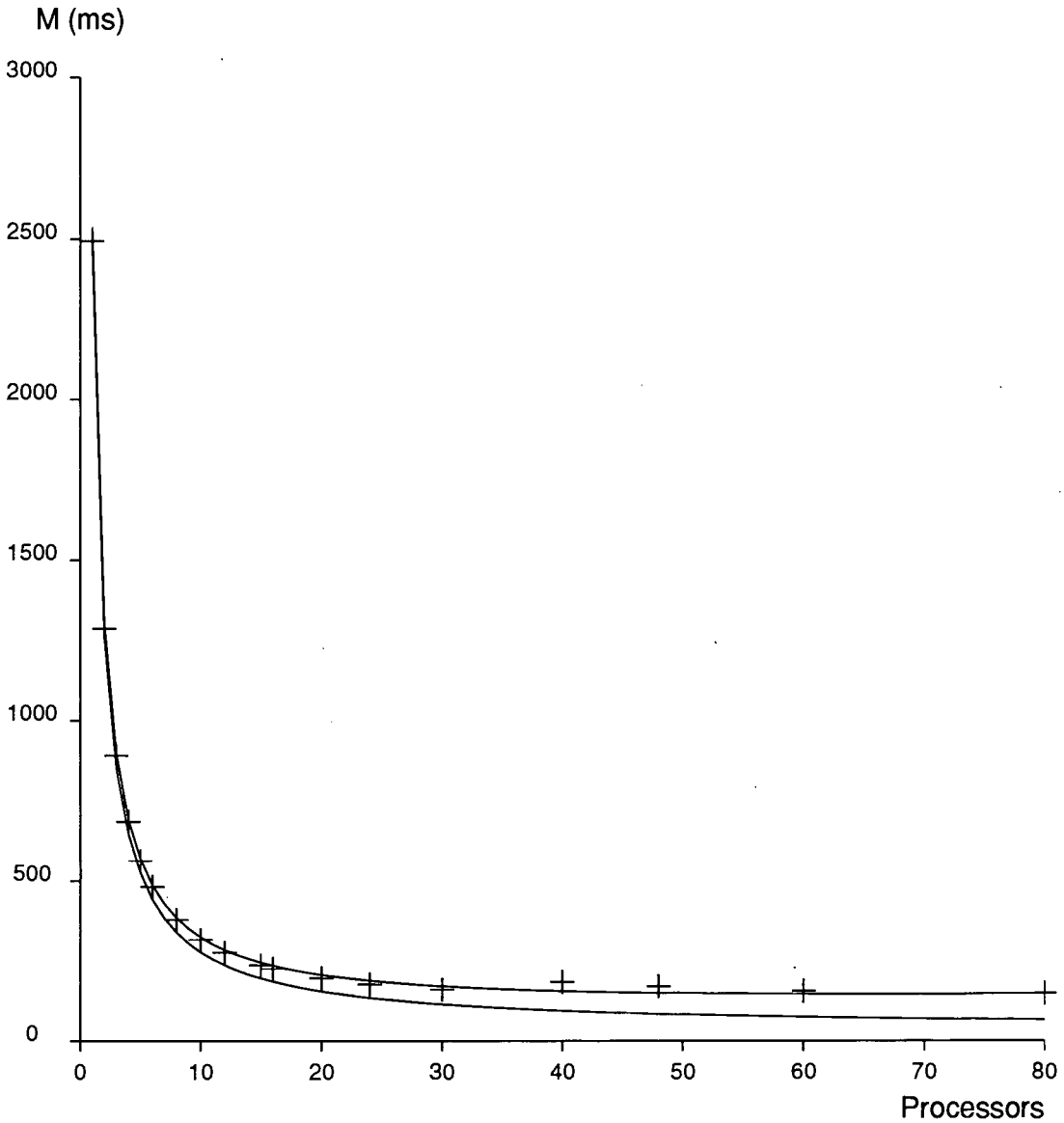


Figure 9–3: Performance with Constant Message Size: Observed (points) *vs* that Predicted by Regression Parameters (line)

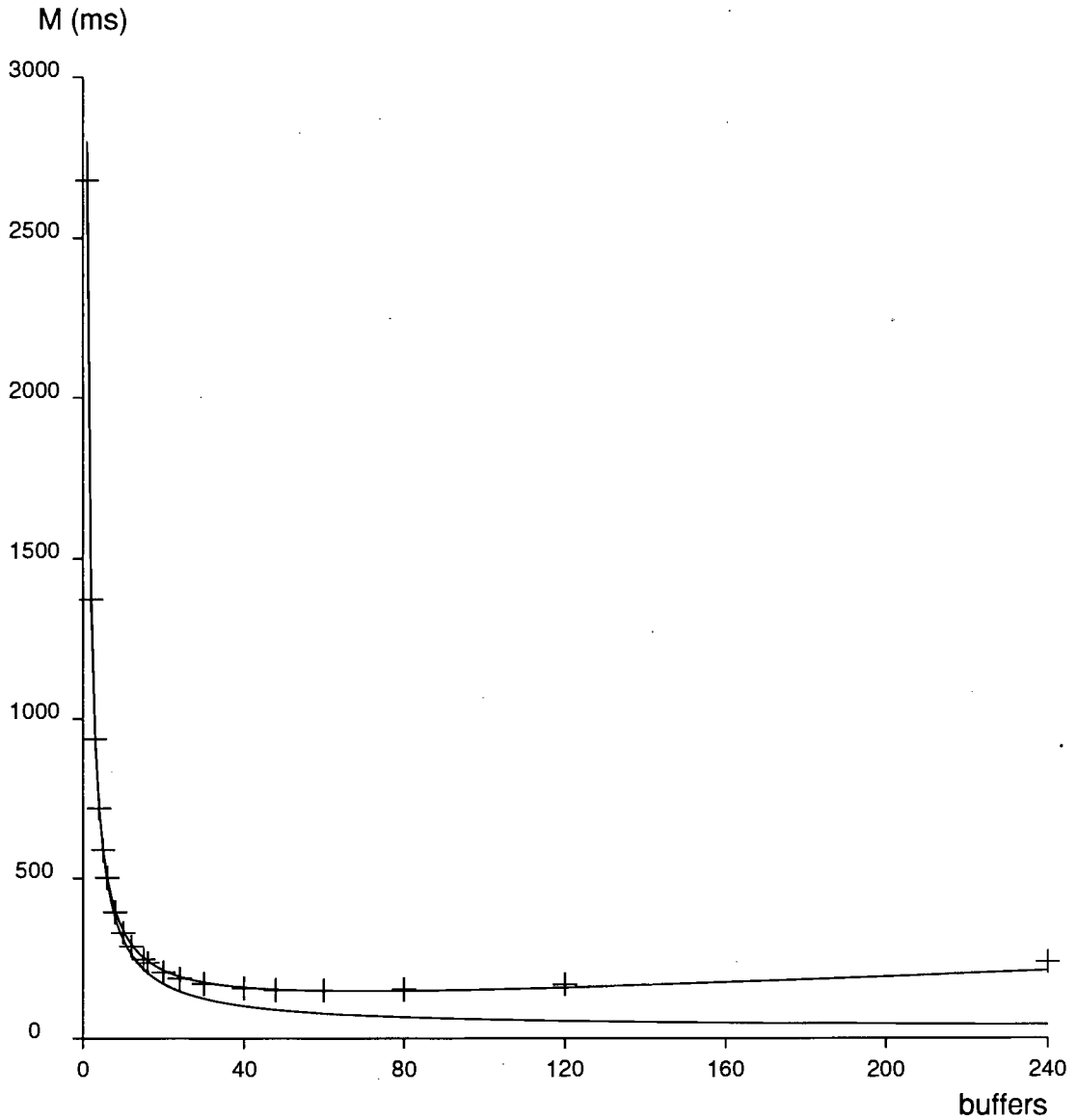


Figure 9–4: Performance with Constant Number of Processors: Observed (points) *vs* that Predicted by Regression Parameters (line)

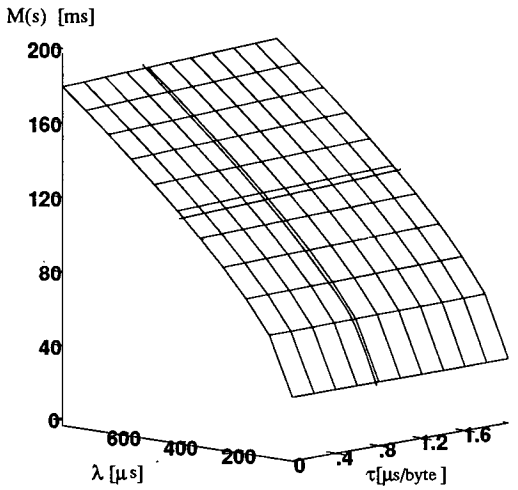


Figure 9–5: Makespan Against $\hat{l}$ and $\hat{\tau}$ for $n = 240, k = 80, \omega = 43.3, m = m_{opt}$

the relationship between M, $\hat{l}$ and $\hat{\tau}$ at optimal Message size,m_{opt}, which we approximate by the expression

$$m_{opt} = \frac{n}{\sqrt{l}}\sqrt{1 + \tau\frac{k}{n}}$$

The result is plotted in Figure 9–5 for $k = 80, n = 240$, in the range $0 < \hat{l} \leq 800$, and $0 \leq \hat{\tau} \leq 2$, which includes the value $\hat{l} = 475$, $\hat{\tau} = 0.775$ as measured for CS-Tools at the intersection of the two cross-hair like lines.

Consider a version of CS-tools with *infinite bandwidth*, that is with $\hat{\tau} = 0$, by moving the crosshairs along the line parallel to the $\hat{\tau}$ axis to the point where it intersects the $(\hat{l}, M)$ plane. It is clear that the line is almost horizontal: there is very little to be gained from increasing bandwidth.

Now move back to the point marked by the cross-hairs for CS-tools and move along the line parallel to the $\hat{l}$ axis to the point where it intersects the $(\hat{\tau}, M)$ plane. This corresponds to CS-tools with the same bandwidth as normal but *zero overhead*. This line is *not* horizontal: it dips steeply down towards the end with zero overhead, indicating that in the case of this numerical computation

substantial gains in performance can be achieved by moving to a message passing system with lower overhead.

The figure shows values expressed in real units ie. not normalised with respect to node calculation time. If the processor was ten times as fast, we can use the approximation that it would compute all sequential computations at ten times the speed, and divide the numbers on each axis by ten to get the performance curve for message passing systems on multicomputers whose processors run ten times faster, say for the Intel iPSC/860 [Intel 1991].

9.4 The Claud Model and our Multicomputer Computation

It is instructive to consider how the Claud model corresponds to the way computation and communication are effected in a CS-tools program, and thus the properties of the diamond dag and of our implementation which may lead to our assumptions being accurate in this special case but not necessarily so in the case of a general computation.

CS-tools on transputer-based communications hardware is basically a store and forward network, but one where the item which is stored and forwarded is not a packet but a subcomponent of a packet known as a flit. The technique is sometimes known as virtual cut-through [Kermani and Kleinrock, 1979]. The actual transfer of data between processors is accomplished by direct memory access units which do not interrupt computation on the processor (except by competing for memory bandwidth). *Software* overheads are, however, incurred at source and destination processors, and also at intermediate processors in the processor network through which messages may be forwarded. We can

assume that in the case of synchronous transfer, communication is effected in the following way.

The message is transferred in packets fragmented into 32 byte flits, the first of which contains routing information. These packets are by default 1Kbyte long. The head packet of a message is sent at the point at which the `csn_tx` call is made. If there is message space available for storage of the packet at the destination an acknowledgement is sent back to the source processor, which then sends the next packet. In quiet networks across small numbers of interprocessor links the acknowledgement can be received at the source before it has transmitted the last flit of the packet to which it refers, and so message transfer can continue as an uninterrupted stream of flits (from consecutive packets) until the last packet has been transferred, or until no message space is available at the destination.

Message space is always available if the receiving processor has made the `csn_rx` call at the time at which the flit arrives: the message is simply copied directly to the address specified in the `csn_rx` call. If the message is too large, the tail is discarded and an error condition is raised. If, however, the receive call has not yet been made when the first flit arrives, the first few packets of the message can be buffered at destination in CS-tools' internal buffering, and then copied out to the address specified in the `csn_rx` call. Only if CS-tools buffering (which is of an unspecified size) is exhausted will the flow of flits be staunched, because in this case a failure token is returned to the sending processor instead of an acknowledgement. At some unspecified time after receiving the failure token the sending processor will retry sending the failed packet.

9.4.1 Matching CS-tools to our Model

It is less than clear that our three parameters: $\hat{l}_s$ and $\hat{l}_r$ (the lengths of atomic computations required to send and to receive a message of arbitrary length) and $\hat{\tau}$, a latency per unit length of message being sent, are adequate to model such

complex behaviour. The following properties of our diamond dag software may be improving the fit of our model.

1. Messages are only sent between adjacent processors along interprocessor links which are not used for any other messages for the duration of the computation.

2. Non-Blocking send calls are used.

3. Messages are short: at most of length $O(n)$ which, in the case of our test example, works out at 240 units of length 8 bytes – or 2 packets.

As a result of 1, we do not see the computation overheads of or the latency associated with the store and forward of flits. In addition there is no contention between messages for interprocessor communications links.

As a result of 2, the sending process does not wait until a message has arrived before it continues computing other nodes. This means that it is indeed appropriate to consider $\hat{\tau}$ as a delay associated with satisfying the precedence relation rather than a cost at the sending processor.

As a result of 3, CS-tools internal buffering will render it unlikely that failure tokens and retries will occur even if there is some mismatch between the timing of messages being sent and `csn_rx` calls being made.

There still remain two glaring problems with our model. First $\hat{l}_s$ and $\hat{l}_r$ contain no *per-flit* and *per-message* computation overhead. This means that they are underestimates of the cost of sending anything other than the unit length messages for which they were measured. Their underestimation grows as message size grows. Second, $\hat{\tau}$ contains no component for the flit of each packet that contains its header, so it is an underestimate for anything other than the asymptotically large messages for which it was measured. Even if this is taken

into account, the fixed flit size means that $\hat{\tau}$ is only accurate for packets of an exact number of flits, and messages of an exact number of packets.

9.4.2 Atomicity of Communications Events

A third problem is more subtle. It is important to notice that the diamond dag software does not specify a schedule; it specifies a partition of the node set; a message set; and a complete order of the node computations and `csn_rx` `csn_txnb` and `csn_test` calls at any given processor. Computation is performed according to the order specified, as and when preceding computations are completed and messages are received. The exact timing of events is the same as computed by our schedule if the measured parameters are accurate, and if the order of computation events is preserved. The problem lies in the way in which `csn_txnb` and `csn_rx` calls correspond to computation events at the destination processor, even if we assume there are no *per-flit* overheads. The problem is outlined below for the receiving processor. There are analogous problems for the sending processor when the computation associated with message protocol and `csn_test` calls is taken into consideration.

If we consider the Gantt charts in Figure 9–6, Processor X sends a message of length η to processor Y at time t. Processor Y has received it at time $t + \eta\hat{\tau}$. According to our model this engenders a computation event of length $\hat{l}_r$ at time $t + \eta\hat{\tau}$. In practice, as shown in the Gantt charts in Figure 9–6, there may well be some component of overhead incurred when the `csn_r<x` call is made (A), some when the message arrives (B), and some when the `csn_rx` call completes (C).

The sequence of events on processor X could be as shown in scenario 1. It makes a `csn_rx` call (overhead A) at time $s < t$. It waits. The first flit of the message arrives and overhead B incurred. When all flits have arrived the `csn_rx` call completes and overhead C occurs. Alternatively, as shown in scenario Y, the first flit of the message arrives unexpectedly at processor Y and overhead

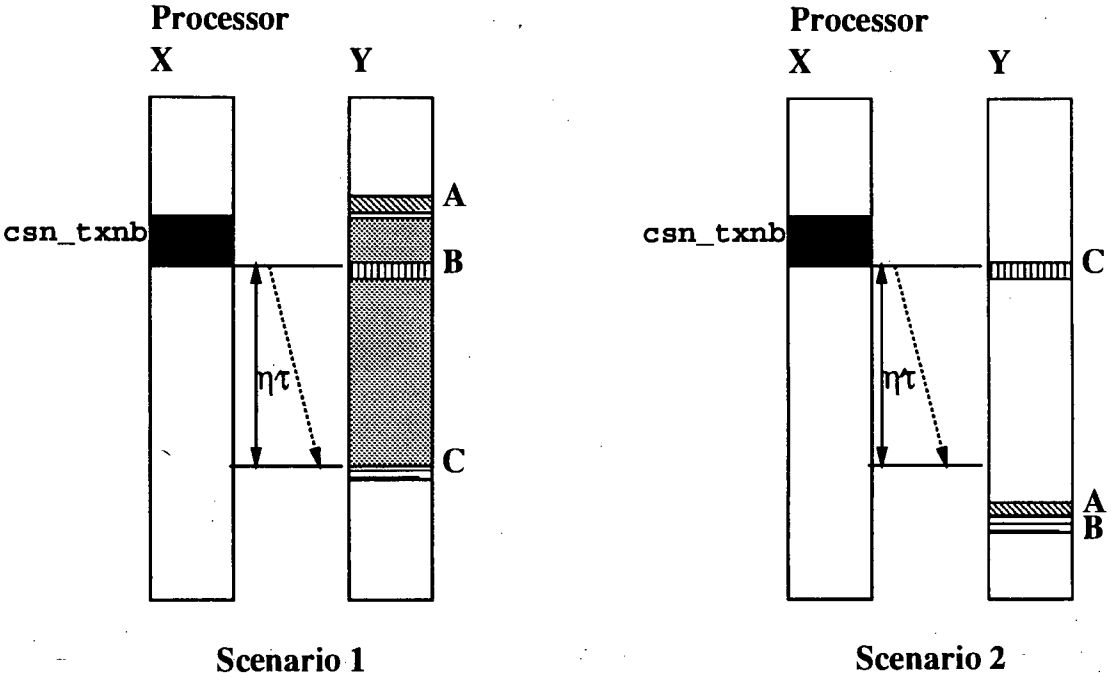


Figure 9–6: Gantt Charts Illustrating Overheads that may be Attributable to $\hat{l}_r$

B is incurred. At time $u > t$ the `csn_rx` call is initiated overhead A is incurred. Then immediately afterwards it completes and overhead C is incurred.

Overhead B is guaranteed to occur at a time greater than t. Overhead C is guaranteed to occur at a time greater than $t + \eta\hat{\tau}$. Nothing can be said about the timing of overhead A, except with reference to the local task schedule at processor Y. These three overheads may vary in magnitude according to the relative timings of the message receipt and the `csn_rx` call. The value of $\hat{l}_r$ that was measured was the sum of all three overheads for the case when the message was received before the `csn_rx` call was made. There are clearly a number of approximations involved in considering $\hat{l}_r$ as a single overhead incurred at time $t + \eta\hat{\tau}$. Some of these may be being masked by the lock-step nature of the computation in the equal message partition of the diamond dag.

Chapter 10

Concluding Remarks

10.1 A Summary of Results

Below is a summary of the results in the thesis. It is a selective summary. For example, some theorems and lemmas are not mentioned, or are mentioned in a specific context when they have more general utility.

10.1.1 Models and the Framework

Throughout the thesis there have been attempts to describe the models underlying formulations of the mapping problem in terms of the unifying framework of Chapter 1.3. At this point we wish to highlight the way in which the models differ in terms of the components of the framework they incorporate. Table 10–1 gives a summary of the non-zero terms in tables 3–1, 3–2, 3–3, 5–2, 5–3 and 7–1.

10.1.2 Complexity

There are four complexity results in the thesis. Two of them refer to scheduling problems, a third to mapping in process based models, and the fourth to the remapping of schedules.

Scheduling problems tend to be NP complete if the underlying models are non-UET. We showed that scheduling in the 2-processor 1&2 unit execution time model is NP complete with unit interprocessor communication delay (Theorem 3.7). This is not altogether surprising since it is NP complete with no communication delay. More interesting, perhaps, is the result (Theorem 6.2) that two processor UET scheduling is NP complete if communications contention is introduced. It is not known whether there is any fixed value of x such that x processor scheduling is NP complete in models with contention-free communication delay or without communication delay.

State		Model					
		Scheduling Models					
		No Communication	No Delay	General Delay	Process Based	Graph Matching	Claud
T_{Calc}		√	√	√	√	√	√
T_{Comm}	T_{Init}						√
	T_{Term}						√
	T_{Route}						
T_{House}	T_{Recalc}			√			√
	T_{Inter}						
	T_{Sched}						
T_{Idle}	T_{Wait_S}		√	√	√		√
	T_{Wait_D}			√	√	√	√
	T_{Fin}	√	√	√	√		√

Table 10–1: A Summary of the Presence of Components of our Framework in Models

The complexity of *bijective* mapping in graph matching models is open. It is related to graph isomorphism. We showed the NP completeness of bijective mapping in graph matching models where the processor graph is the variable to be optimised (Theorem 6.3).

Papadimitriou and Yannakakis give an approximation algorithm for scheduling in models with communication delay which is guaranteed to find a mapping with a performance within a factor of two of optimal—at least in terms of the model. We showed that there still remains an NP complete problem if a multicomputer programmer wishes to use such a schedule on a multicomputer with a fixed number of processors (Theorem 4.2).

10.1.3 Bounds on Processor Usage

We derived a number of bounds beyond which there are no performance gains associated with introducing more processors into a scheduling model. We can refer to the number of processors beyond which performance benefits can be achieved as P_{max}. In Table 10–2 we give the results we have shown for P_{max} in various scheduling models. We refer to a dag $\Lambda = (\Gamma, \Delta)$.

10.1.4 Performance Guarantees of Algorithms

We considered remapping a PNY schedule to a fixed number of processors for comparison with ETF. The performance bounds for the algorithms are similar, although the PNY bound is better for low values of τ and when the number of processors is high in comparison to the width of the dag. The ETF bound is better for high values of τ and when the number of processors is low in comparison to the width of the dag. We showed that on some dags an ETF schedule can be a factor of τ worse than a remapped PNY schedule, and on some dags it can be a factor of τ better.

Model	Result	Reference
No Communication UET	$\forall\Lambda, P_{max} = \|\Gamma\|$	Theorem 3.4 and Corollary 3.1
No Communication	$\forall\Lambda, P_{max} \leq \|\Gamma\|$	Corollary 3.1
No Delay	$\forall\Lambda, P_{max} \leq W\|\Lambda\|$	Theorem 3.3
Unit Delay No Replication	$\forall\Lambda, P_{max} \leq W\|\Lambda\|$	Theorem 3.5
Unit Delay UET	$\exists\Lambda : P_{max} > W\|\Lambda\|$	Section 3.2.3
Uniform Delay	$\forall\Lambda, P_{max} \leq \|\Gamma\|$	Corollary 3.2
General Delay UET	$\exists\Lambda : P_{max} > \|\Gamma\|$	Theorem 3.6

Table 10–2: A summary of Bounds Beyond Which There is No Performance to be Gained by Introducing Extra Processors

Now we consider mapping a UET *Fixed Delay* scheduling model with a communication delay of at most τ. When we refer to "optimal makespan" below, we mean the optimal makespan of schedules on a number of processors equal to the number of tasks. In the case of both PNY and ETF we can generate performance bounds which are independent of the *width* of the dag on a number of processors which is a linear function of the width of the dag. In the case of PNY we simply use Algorithm 4.3, apply Algorithm 3.2 and, by Theorem 4.5, we end up with a schedule with makespan at most *twice* optimal for which the processor usage is at most $(\tau + 1)/2$ times the width of the dag. In the case of ETF our performance guarantee of Section 4.4.1 states that we will get within a factor of $\tau + 2$ of optimal makespan if we use as many processors as the width of the dag. In Section 7.2.3 we showed such performance guarantees are not available for Claud models, even for those that are *No Delay Fixed Cost* and UET.

10.1.5 Nature of Predictions

There are a number of different results in Chapter 8. An interesting way to summarise some of them is to consider the plight of a multicomputer programmer who is writing a longest common subsequence program and trying to decide whether to write a program corresponding to the lines schedule or to the stripes schedule. If she believed that *event cost* and *chain cost* had predictive power for her multicomputer, she would use the stripes schedule in preference to the lines schedule. If she believed that the *Fixed Delay* scheduling model had predictive power then she would use a lines schedule.

Another interesting prediction which appears in Chapters 8 and 9 is that in *Fixed Delay* scheduling models and in *Fixed Delay No Cost* Claud models, adding processors to the model can never extend the makespans of stripes schedules of the diamond dag. In *Fixed Delay Fixed Cost* Claud models there is an optimum number of processors.

10.1.6 Predictive Power

It appears that for a program corresponding to our diamond dag and parameters derived independently from the target multicomputer, the predictive power of a Claud model is very high. Indeed, with a high degree of accuracy, we are able to use the performance of the computation to predict the parameters of multicomputer. Moreover, there is indeed an optimum number of processors for a given size of a computation which corresponds to the diamond dag, and that optimum is roughly where we predict it to be.

10.2 Verisimilitude, Complexity and Predictive Power

The word *verisimilitude* appears once previously in this thesis. It means "apparent increase in believability" and embodies a useful concept for discussing the models underlying mapping problem. One can formulate mapping problems in terms of models which have features corresponding to more and more of the properties of the multicomputer. This may lead to an increase in the number of non-zero terms when the models are related to our framework. Alternatively, it may lead to an apparent modelling of more features of the interprocessor communications medium. In short it may increase the verisimilitude of the model.

The problem with added verisimilitude is that it leads to added complexity: added computational complexity, non-approximability and excessively complicated notation.

On the other hand, models with different levels of verisimilitude give different predictions and lead the multicomputer programmer in different directions. If progress is to be achieved in the mapping problem for multicomputers, models must be developed with the lowest level of verisimilitude that can achieve predictive power. Fusion of Claud with a model incorporating message contention may well be a good way forward.

Bibliography

[Agrawal and Jagadish, 1988] Agrawal, R. and Jagadish, H. (1988). Partitioning techniques for large grain parallelism. *IEEE Trans. Comput.*, C-37(12):1627–1634.

[Aho et al., 1976] Aho, A., Hirschberg, D., and Ullman, J. (1976). Bounds on the complexity of the longest common subsequence problem. *Journal of the Association for Computing Machinery*, 23(1):1–12.

[Aho et al., 1983] Aho, A., Hopcroft, J., and Ullman, J. (1983). *Data Structures and Algorithms*. Addison Wesley.

[Al-Mouhammed, 1990] Al-Mouhammed, M. (1990). Lower bound on the number of processors and time for scheduling precedence graphs with communication costs. *IEEE Trans. Software Engrg.*, SE-16(12):1390–1301.

[Baccelli and Liu, 1990] Baccelli, F. and Liu, Z. (1990). On the execution of parallel programs on multiprocessor systems — a queueing theory approach. *J. ACM*, 37(2):373–414.

[Bal et al., 1989] Bal, H., Stenier, J., and Tanenbaum, A. (1989). Programming languages for distributed computing systems. *Computing Surveys*, 21(3):261–322.

[Baxter and Patel, 1989] Baxter, J. and Patel, J. (1989). The LAST algorithm: A heuristic based static task allocation algorithm. In *Proc. Intl. Conf. Parallel Comput.*, volume 2, pages 217–222.

[Beauwens, 1989] Beauwens, R. (1989). Approximate factorisations with S.P.C.O M-factors. *Bit*, 29:658–681.

[Berger and Cowen, 1991] Berger, B. and Cowen, L. (1991). Complexity results for $\{<, \leq, =\}$-constrained scheduling. In *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 137–147.

[Berman and Snyder, 1987] Berman, F. and Snyder, L. (1987). On mapping parallel algorithms into parallel architectures. *J. Parallel Dist. Comput.*, 4:439–458.

[Blazewicz et al., 1986] Blazewicz, J., Drabowski, M., and Weglarz, J. (1986). Scheduling multiprocessor tasks to minimize schedule length. *IEEE Trans. Comput.*, C-35(5):389–393.

[Blazewicz et al., 1991] Blazewicz, J., Dror, M., and Weglarz, J. (1991). Mathematical programming formulations for machine scheduling: A survey. *European Journal of Operational Research*, 51:283–300.

[Blazewicz et al., 1984] Blazewicz, J., Weglarz, J., and Drabowski, M. (1984). Scheduling independent 2-processor tasks to minimise schedule length. *Inform Process. Lett.*, 18(5):267–273.

[Bokhari, 1981a] Bokhari, S. (1981a). On the mapping problem. *IEEE Trans. Comput.*, C-30(3):207–214.

[Bokhari, 1981b] Bokhari, S. (1981b). A shortest tree algorithm for optimal assignments across space and time in a distributed computer system. *IEEE Trans. Software Engrg.*, SE-7(6):583–589.

[Bokhari, 1988] Bokhari, S. (1988). Partitioning problems in parallel, pipelined and distributed computing. *IEEE Trans. Comput.*, C-37(1):48–57.

[Bokhari, 1990] Bokhari, S. (1990). Communication overhead on the Intel iPSC-860 Hypercube. Technical Report 10, ICASE, NASA, Langley Research Center, Virginia. ICASE Interim Report.

[Bomans and Roose, 1989] Bomans, L. and Roose, D. (1989). Benchmarking the iPSC/2 Hypercube multiprocessor. *Concurrency Practice and Experience*, 1(1):3–18.

[Brent, 1974] Brent, R. (1974). The parallel evaluation of general arithmetic expressions. *J. ACM*, 21:201–206.

[Bruno et al., 1974] Bruno, J., Coffman Jr., E., and Sethi, R. (1974). Scheduling independent taskms to reduce mean finishing time. *Comm. ACM*, 17(7):382–387.

[Casavant and Kuhl, 1988] Casavant, T. and Kuhl, J. (1988). A taxonomy of scheduling in general purpose distributed computing systems. *IEEE Trans. Software Engrg.*, SE-14(2):141–154.

[Chen and Lai, 1988] Chen, G.-I. and Lai, T.-H. (1988). Scheduling independent jobs on hypercubes. In Cori, R. and Wirsing, M., editors, *Proc Conf. Theoretical Aspects of Computer Science*, pages 273–280.

[Chen and Shin, 1987] Chen, M.-S. and Shin, K. (1987). Processor allocation in a n-cube multiprocessor using gray codes. *IEEE Trans. Comput.*, C-36(12):1396–1407.

[Chen and Liu, 1975] Chen, N. and Liu, C. (1975). On a class of scheduling algorithms for multiprocessor computing systems. In Feng, T.-Y., editor, *Lecture Notes in Computer Science*. Springer, New York.

[Cheng and Sin, 1990] Cheng, T. and Sin, C. (1990). A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research*, 47:61–63.

[Chittor and Enbody, 1992] Chittor, S. and Enbody, R. (1992). Minimising Contention: A New Mapping Objective for Second Generation Multicomputers. Technical report, University Michigan State University, East Lansing, MI 48824-1027, USA. Submitted to IEEE Transactions on Parallel and Distributed Systems.

[Cho and Sahni, 1980] Cho, Y. and Sahni, S. (1980). Bounds for list schedules on uniform processors. *SIAM J. Comput.*, 9(1):91–103.

[Chrétienne, 1989] Chrétienne, P. (1989). A polynomial algorithm to optimally schedule tasks on a virtual distributed system under tree-like precedence constraints. *European J. Oper. Res.*, 43:225–230.

[Chrétienne and Picouleau, 1992] Chrétienne, P. and Picouleau, C. (1992). The basic scheduling problem with interprocessor communication delays. In *Proceedings of the Summer School on Scheduling Theory and its Applications*, Chateau de Bonas, France.

[Chu et al., 1984] Chu, L., Lan, M.-T., and Hellerstein, J. (1984). Estimation of intermodule communication (IMC) and its applications in distributed processing systems. *IEEE Trans. Comput.*, C-33(8):691–699.

[Chu et al., 1980] Chu, W., Holloway, L., Lan, M.-T., and Efe, K. (1980). Task allocation in distributed data processing. *Computer*, 13(11).

[Chu and Lan, 1987] Chu, W. and Lan, M.-T. (1987). Task allocation and precedence relations for distributed real-time systems. *IEEE Trans. Comput.*, C-36(6):667–679.

[Coffman Jr., 1976] Coffman Jr., E., editor (1976). *Computer and Job Shop Scheduling Theory*. John Wiley, New York.

[Coffman Jr. et al., 1984a] Coffman Jr., E., Flatto, L., and Leuker, G. (1984a). Expected makespans for largest-first multiprocessor scheduling. In Gelenbe, E., editor, *Performance '84*, pages 491–506. Elsevier Science Publishers B.V. (North Holland).

[Coffman Jr. et al., 1978] Coffman Jr., E., Garey, M., and Johnston, D. (1978). An application of bin-packing to multiprocessor scheduling. *SIAM J. Comput.*, 7(1):1–17.

[Coffman Jr. et al., 1984b] Coffman Jr., E., Garey, M., and Johnston, D. (1984b). Approximation algorithms for bin-packing – an updated survey. In Ausellio, G., Lucertini, M., and Serafini, P., editors, *Algorithm Design for Computer System Design*, Berlin. Springer-Verlag.

[Coffman Jr. and Graham, 1972] Coffman Jr., E. and Graham, R. (1972). Optimal scheduling for two processor systems. *Acta Informatica*, 1:200–213.

[Cole and Vishkin, 1988] Cole, R. and Vishkin, U. (1988). Approximate parallel scheduling, part 1: The basic technique with applications to optimal parallel list ranking in logarithmic time. *SIAM J. Comput.*, 17(1):128–142.

[Conway et al., 1967] Conway, R., Maxwell, W., and Miller, L. (1967). *Theory of scheduling*. Addison-Wesley, Reading, Mass.

[Cvetanovic, 1987] Cvetanovic, Z. (1987). The effects of problem partitioning, allocation and granularity on the performance of multiple-processor systems. *IEEE Trans. Comput.*, C-36(4):421–432.

[Dally, 1990a] Dally, W. (1990a). Network and processor architecture for message-driven computers. In Suaya, R. and Birtwhistle, G., editors, *VLSI and Parallel Computation*, pages 140–222. Morgan Kaufmann, Palo Alto, CA.

[Dally, 1990b] Dally, W. (1990b). Performance analysis of k-ary n-cube interconnection networks. *IEEE Transactions on Computers*, 39:775–785.

[Dally and Seitz, 1987] Dally, W. and Seitz, C. (1987). Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Trans. Comput.*, C-36(5).

[Dietz et al., 1992] Dietz, H., Zaafrani, A., and O'Keefe, M. (1992). Static scheduling for barrier MIMD architectures. *The Journal of Supercomputing*, 5,4:263–289.

[Du and Leung, 1989] Du, J. and Leung, J.-T. (1989). Complexity of scheduling parallel task systems. *SIAM J. Disc. Math*, 2(4):473–487.

[Duato, 1992] Duato, J. (1992). Impact of locality on the performance of some adaptive routing algorithms for the hypercube. In Joosen, W. and Milgrom, E., editors, *Parallel Computing: from Theory to Sound Practice*. IOS Press.

[Efe, 1982] Efe, K. (1982). Heuristic models of task assignment scheduling in distributed systems. *IEEE Computer*, June 1982:50–56.

[El-Rewini and Lewis, 1990] El-Rewini, H. and Lewis, T. (1990). Scheduling parallel program tasks onto arbitrary target machines. *J. Parallel Dist. Comput.*, 9:138–153.

[Fellows and Langston, 1988] Fellows, M. and Langston, M. (1988). Processor utilisation in a linearly connected parallel processing system. *IEEE Trans. Comput.*, C-37(5):594–603.

[Fernández and Bussell, 1973] Fernández, E. and Bussell, B. (1973). Bounds on the number of processors and time for multiprocessor optimal schedules. *IEEE Trans. Comput.*, C-22(8):745–751.

[Fernández-Baca, 1989] Fernández-Baca, D. (1989). Allocating modules to processors in a distributed system. *IEEE Trans. Software Engrg.*, SE-15(11):1427–143.

[Fox, 1989] Fox, G. (1989). Parallel computing comes of age. *Concurrency Practice and Experience*, 1(1):63–103.

[Fox et al., 1988] Fox, G., Johnson, M., Lyzenga, G., Otto, S., Salmon, J., and Walker, D. (1988). *Solving Scientific Problems on Concurrent Processors.* Prentice Hall, New Jersey.

[Gabow, 1988] Gabow, H. (1988). Scheduling UET systems on two uniform processors and length two pipelines. *SIAM J. Comput.*, 17(4):810–811.

[Garey et al., 1977] Garey, M., Graham, R., and Johnston, D. (1977). Performance guarantees for scheduling algorithms. *Operations Research*, 26(1).

[Garey and Johnson, 1975] Garey, M. and Johnson, D. (1975). Complexity results for multiprocessor scheduling with resource constraints. *SIAM J. Comput.*, 4(4):396–411.

[Garey and Johnson, 1979] Garey, M. and Johnson, D. (1979). *Computers and Intractability.* W.H. Freeman and Co., San Francisco.

[Garey et al., 1976] Garey, M., Johnson, D., and Stockmeyer, L. (1976). Some simplified NP-complete graph problems. *Theor. Comput. Sci.*, 1:237–267.

[Garey and Johnson, 1982] Garey, M. and Johnson, R. (1982). Approximation algorithms for bin packing problems – a survey. In G. Ausellio, M., ed-

itor, *Analysis and Design of Combinatorial optimisation*, pages 147–172. Springer Verlag, Vienna, Austria.

[Gaudiot et al., 1988] Gaudiot, J., Pi, J., and Campbell, M. (1988). Program graph allocation in distributed multicomputers. *Parallel Computing*, 7:227–247.

[Gonzalez, 1977] Gonzalez, M. (1977). Deterministic processor scheduling. *Computing Surveys*, 9(3).

[Graham, 1966] Graham, R. (1966). Bounds for certain multprocessing timing anomalies. *Bell System Technical J.*, 45:1563 – 1581.

[Graham, 1969] Graham, R. (1969). Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math.*, 17(2):416–429.

[Graham et al., 1979] Graham, R., Lawler, E., Lenstra, J., and Rinnooy Kan, A. (1979). Optimisation and approximation in deterministic sequencing and scheduling a survey. *Annals of Discrete Mathematics*, 5:287–236.

[Gusfield, 1983] Gusfield, D. (1983). Parametric combinatorial computing and a problem of program module distribution. *J. ACM*, 30(3):551–563.

[Gylys and Edwards, 1976] Gylys, V. and Edwards, J. (1976). Optimal partitioning of workload for distributed systems. In *Proc. Compcon Fall 76.*

[Hageman and Young, 1981] Hageman, L. and Young, D. (1981). *Applied iterative methods.* Academic Press.

[Helmbold and Mayr, 1987] Helmbold, D. and Mayr, E. (1987). Two processor scheduling is in NC. *SIAM J. Comput.*, 16(4):747–759.

[Hochbaum and Shmoys, 1988a] Hochbaum, D. and Shmoys, D. (1988a). A polynomial approximation scheme for scheduling on uniform processors using the dual approximation approach. *SIAM J. Comput.*, 17(3):539–551.

[Hochbaum and Shmoys, 1988b] Hochbaum, D. and Shmoys, D. (1988b). Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *J. ACM*, 34(1):144–162.

[Houstis, 1990] Houstis, C. (1990). Module allocation of real-time applications to distributed systems. *IEEE Trans. Software Engrg.*, SE-16(7):699–709.

[Hu, 1961] Hu, T. (1961). Parallel sequencing and assembly line problems. *Oper Res.*, 9:841–848.

[Hwang et al., 1989] Hwang, J.-J., Chow, Y.-C., Anger, F., and Lee, C.-Y. (1989). Scheduling precedence graphs in systems with interprocessor communication times. *SIAM J. Comput.*, 18(2):244–257.

[IEEE, 1991] IEEE (1991). SCI, scalable coherent interconnect. Technical Report P1596/D1.98, IEEE Standards Department, 345 East 47th Street, New York, N.Y. 1007 USA. Draft for Review by Negative Ballot Review Committee.

[Indurkhya and Stone, 1986] Indurkhya, B. and Stone, H. (1986). Optimal partitioning of randomly generated parallel programs. *IEEE Trans. Software Engrg.*, SE-12(3):483–495.

[INMOS Limited, 1988] INMOS Limited (1988). *The Transputer Reference Manual*. Prentice Hall.

[Intel, 1991] Intel (1991). ipsc/860 parallel supercomputer product overview. Technical report, Intel Corporation Supercomputer Systems Division.

[Johnsson, 1990] Johnsson, S. (1990). Communication in network architectures. In Suaya, R. and Birtwhistle, G., editors, *VLSI and Parallel Computation*, pages 223–378. Morgan Kaufmann, Palo Alto, CA.

[Jung et al., 1989] Jung, H., Kirousis, L., and Spirakis, P. (1989). Lower bounds and efficient algorithms for multiprocessor scheduling of dags with communication delays. In *Proc. ACM Symposium on Parallel Algorithms and Architectures*, pages 254–264.

[Kafura and Shen, 1977] Kafura, D. and Shen, V. (1977). Task scheduling on a multiprocessor system with independent memories. *SIAM J. Comput.*, 6(1):167–187.

[Kapelnikov et al., 1989] Kapelnikov, A., Muntz, R., and Ercegovac, M. (1989). A modelling methodology for the anlaysis of concurrent systems and computations. *J. Parallel Dist. Comput.*, 6:568–597.

[Kaufmann, 1974] Kaufmann, M. (1974). An almost-optimal algorithm for the assembly line scheduling problem. *IEEE Trans. Comput.*, C-23(11):1169–1174.

[Kermani and Kleinrock, 1979] Kermani, P. and Kleinrock, L. (1979). Virtual cut-through: A new computer communication switching technique. *Computer Networks*, 3:287–286.

[Kim and Browne, 1988] Kim, S. and Browne, J. (1988). A general approach to mapping of parallel computations upon multiprocessor architectures. In *International Conference on Parallel Processing* 3, pages 1–8.

[Krämer and Mühlenbein, 1989] Krämer, O. and Mühlenbein, H. (1989). Mapping strategies in message-based multiprocessor systems. *Parallel Computing*, 9:213–225.

[Krishnamurthy, 1990] Krishnamurthy, S. (1990). A brief survey of papers on scheduling for pipelined processors. *Sigplan Notices*, 25(7):97–106.

[Kruatrachue and Lewis, 1988] Kruatrachue, B. and Lewis, T. (January 1988). Grain size determination for parallel programming. *IEEE Software*, pages 23–32.

[Lam and Sethi, 1977] Lam, S. and Sethi, R. (1977). Worst case analysis of two scheduling algorithms. *SIAM J. Comput.*, 6(3):518–536.

[Lamport, 1979] Lamport, L. (1979). How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs. *IEEE Transactions on Computers*, C-28:690–691.

[Lawler, 1982] Lawler, E. (1982). Preemptive scehduling of precedence constrained jobs on parallel machines. In Dempster, M., editor, *Deterministic and Stochastic Scheduling*. D.Reidel Publishing Co.

[Lee and Aggarwal, 1987] Lee, S. and Aggarwal, J. (1987). A mapping strategy for parallel computing. *IEEE Trans. Comput.*, C-36(4):433–442.

[Leighton, 1992] Leighton, F. (1992). *Introduction to Parallel Algorithms and Architectures*. Morgan Kaufmann, San Mateo, CA.

[Lo, 1988] Lo, V. (1988). Heuristic algorithms for task assignment in distributed systems. *IEEE Trans. Comput.*, 37(11):1384–1397.

[Lo, 1992] Lo, V. (1992). Temporal Communication Graphs: Lamport's Process-Time Graphs Augmented for the Purpose of Mapping and Scheduling. Technical Report CIS-TR-92-05, University of Oregon, Eugene, Oregon 97403. To appear in Journal of Parallel and Distributed Computing.

[Ma et al., 1982] Ma, P.-Y., Lee, E., and Tsuchiya, M. (1982). A task allocation model for distributed computing systems. *IEEE Trans. Comput.*, C-31(1):246–252.

[Mak and Lundstrom, 1990] Mak, V. and Lundstrom, S. (1990). Predicting performance of parallel computations. *IEEE Trans. Paral. Distr. Comput.*, 1(3):257–270.

[Manneback et al., 1992] Manneback, P., Quin, J., and Libert, G. (1992). Performance models of MICCG algorithm on distributed memory MIMD computers. In Joosen, W. and Milgrom, E., editors, *Parallel Computing: from Theory to Sound Practice*, pages 52–59. IOS Press.

[Martel, 1988] Martel, C. (1988). A parallel algorithm for preemptive scheduling of uniform machines. *J. Parallel Dist. Comput.*, 5:700–715.

[McDowell and Appelbe, 1986] McDowell, C. and Appelbe, W. (1986). Processor scheduling for linearly connected parallel processors. *IEEE Trans. Comput.*, C-35(7):632–638.

[McGreary and Gill, 1989] McGreary, C. and Gill, H. (1989). Automatic determination of grain size for efficient parallel programming. *Comm. ACM*, 32(9).

[McNaughton, 1959] McNaughton, R. (1959). Scheduling with deadlines and loss functions. *Management Science*, 6:1–12.

[Meiko, 1992] Meiko (1992). So how does the CSN work anyway? Technical report, Meiko Scientific Limited.

[Muntz and Coffman Jr., 1969] Muntz, R. and Coffman Jr., E. (1969). Optimal preemptive scheduling on two-processor systems. *IEEE Trans. Comput.*, C-18(11):1014–1020.

[Nicol, 1989] Nicol, D. (1989). Optimal partitioning of random programs across two processors. *IEEE Trans. Software Engrg.*, SE-15(2):134–141.

[Nicol et al., 1992] Nicol, D., Simha, R., Choudhury, A., and Narahari, B. (1992). Optimal Processor Assignment for Pipeline Computations. Technical Report 91-79, ICASE, NASA, Langley Research Center, Virginia. NASA Contractor Report.

[Norman, 1990] Norman, M. (1990). Multicomputer applications and how to model the mapping problem. In Malek, M., editor, *Parallel Computing for the Physical Sciences*, pages 37–42, Rocqencourt, France. ONR Europe / Inria.

[Norman et al., 1990] Norman, M., Boeres, C., and Thanisch, P. (1990). Minimising message path lengths in multicomputers. Technical Report EPCC-TR-90-17, Edinburgh Parallel Computing Centre. Under Review by Journal of Parallel and Distributed Computing.

[Norman and Thanisch, 1991] Norman, M. and Thanisch, P. (1991). Models of machines and computations for mapping in multicomputers. Technical Report EPCC-TR-92-15, Edinburgh Parallel Computing Centre. Accepted by ACM Computing Surveys.

[Norman and Thanisch, 1993] Norman, M. and Thanisch, P. (1993). Bounds beyond which there are no performance gains from adding processors to scheduling models. Technical Report EPCC-TR-93-06, Edinburgh Parallel Computing Centre. Submitted to Operations Research.

[Norman et al., 1992] Norman, M., Thanisch, P., and Chang, K.-F. (1992). Partitioning dag computations: A cautionary note. In Joosen, W. and Milgrom, E., editors, *Parallel Computing: from Theory to Sound Practice*, pages 360–364. IOS Press.

[Papadimitriou and Ullman, 1987] Papadimitriou, C. and Ullman, J. (1987). A communication-time tradeoff. *SIAM J. Comput.*, 16(4):639–646.

[Papadimitriou and Yannakakis, 1990] Papadimitriou, C. and Yannakakis, M. (1990). Towards an architecture-independent analysis of parallel algorithms. *SIAM J. Comput.*, 19:322–328.

[Picouleau, 1992] Picouleau, C. (1992). New complexity results on the UET-UCT scheduling problem. In *Proceedings of the Summer School on Scheduling Theory and its Applications*, Chateau de Bonas, France.

[Pippenger, 1979] Pippenger, N. (1979). On simultaneous resource bounds (preliminary version). In *Proc. 20th IEEE FOCS*, pages 307–311.

[Pountain, 1989] Pountain, R. (1989). Configuring parallel programs. Part 1. *Byte*, (December):349–352.

[Pritchard et al., 1987] Pritchard, D., Askew, C., Carpenter, D., Glendinning, I., Hey, A., and Nicole, D. (1987). Practical parallelism using transputer arrays. In deBackker, J., Nijman, A., and Treleaven, P., editors, *Parallel Architectures and Languages*, Lecture Notes in Computer Science, page 278. Springer-Verlag, Berlin.

[Ramamritham et al., 1990] Ramamritham, K., Stankovic, J., and Shiah, P.-F. (1990). Efficient scheduling algorithms for multiprocessor systems. *IEEE Trans. Paral. Distr. Comput.*, 1(2):184–194.

[Rao et al., 1979] Rao, G., Stone, H., and Hu, T. (1979). Assignment of tasks in a distributed processor system with limited memory. *IEEE Trans. Comput.*, C-28(4):291–298.

[Rayward-Smith, 1987a] Rayward-Smith, V. (1987a). The complexity of preemptive scheduling given interprocessor communication delays. *Inform. Process. Lett.*, 25(2):123–125.

[Rayward-Smith, 1987b] Rayward-Smith, V. (1987b). UET scheduling with unit interprocessor communication delays. *Discrete Appl. Math.*, 18:55–71.

[Reed and Fujimoto, 1987] Reed, D. and Fujimoto, R. (1987). Multicomputer network operating systems. In *Multicomputer networks : message-based parallel processing*, pages 177–238. MIT Press, Cambridge Mass.

[Sahni, 1976] Sahni, S. (1976). Algorithms for scheduling independent tasks. *J. ACM*, 23(1):116–127.

[Sarkar, 1989] Sarkar, V. (1989). *Partitioning and Scheduling Parallel Programs for Multiprocessors*. Pitman, London.

[Scott et al., 1992] Scott, S., Goodman, J., and Vernon, M. (1992). Performance of the SCI ring. In *Proceeedings of the 19th International Symposium on Computer Architecture*, pages 403–414. IEEE Computer Society Press.

[Seitz, 1990] Seitz, C. (1990). Concurrent architectures. In Suaya, R. and Birtwhistle, G., editors, *VLSI and Parallel Computation*, pages 1–84. Morgan Kaufmann, Palo Alto, CA.

[Sethi, 1976] Sethi, R. (1976). Algorithms for minimal length schedules. In Coffman Jr., E., editor, *Computer and Job Shop Scheduling Theory*. John Wiley, New York.

[Shen and Tsai, 1985] Shen, C.-C. and Tsai, W.-H. (1985). A graph matching approach to optimal task assignment in distributed computing systems using a minimax criterion. *IEEE Trans. Comput.*, C-34(3):197–203.

[Sinclair, 1987] Sinclair, J. (1987). Efficient comutation of optimal assignments for distributed tasks. *J. Parallel Dist. Comput.*, 4:342–362.

[Stone, 1977a] Stone, H. (1977a). Multiprocessor scheduling with the aid of network flow algorithms. *IEEE Trans. Software Engrg.*, SE-3:85–93.

[Stone, 1977b] Stone, H. (1977b). Program assignment in three processor systems and tricutset partitioning of graphs. Technical Report ECE-CS-77-7, Univ. Massachusetts, Amherst.

[Stone, 1978] Stone, H. (1978). Critical load factors in distributed systems. *IEEE Trans. Software Engrg.*, SE-4:254–258.

[Swamy and Thulasiram, 1981] Swamy, M. and Thulasiram, K. (1981). *Graphs, Networks and Algorithms.* John Wiley and Sons.

[Towsley, 1986] Towsley, D. (1986). Allocating programs containing branches and loops within a multiple processor system. *IEEE Trans. Software Engrg.*, SE-12(10):1018–1024.

[Ullman, 1975] Ullman, J. (1975). NP-complete scheduling problems. *J. of Computer and System Sciences*, 10:384–393.

[Upfal, 1984] Upfal, E. (1984). Efficient schemes for parallel communication. *J. ACM*, 31(4).

[Valiant, 1982] Valiant, L. (1982). A scheme for fast parallel communication. *SIAM J. Comput.*, 11:350–361.

[Valiant and Brebner, 1981] Valiant, L. and Brebner, G. (1981). Universal schemes for parallel communication. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing*, pages 263–267. ACM, New York.

[Veltman et al., 1990] Veltman, B., Lageweg, B., and Lenstra, J. (1990). Multiprocessor scheduling with communication delays. *Parallel Computing*, 16:173–182.

[Wachspress, 1966] Wachspress, E. (1966). *Iterative solution of Elliptic systems.* Prentice Hall Int.

[Wang and Cheng, 1992] Wang, Q. and Cheng, K. (1992). A heuristic of scheduling parallel tasks and its analysis. *SIAM J. Comput.*, 21,2:281–294.

[Williams, 1983] Williams, E. (1983). Assigning processes to processors in distributed systems. In *Proceedings IEEE Conference on Parallel Processing*, pages 404–406.

[Yang and Gerasoulis, 1993] Yang and Gerasoulis (1993). List scheduling with and without communication delay. *Parallel Computing*. To Appear.

[Yang and Gerasoulis, 1992] Yang, T. and Gerasoulis, A. (1992). DSC: Scheduling parallel tasks on an unbounded number of processors. Technical report, Rutgers University, Department of Computer Science.